



European Sixth Framework Network of Excellence FP6-2004-IST-026854-NoE

Deliverable D6.4 **Open Source Support and Joint Software Development**

The EMANICS Consortium

Caisse des Dépôts et Consignations, CDC, France
Institut National de Recherche en Informatique et Automatique, INRIA, France
University of Twente, UT, The Netherlands
Imperial College, IC, UK
Jacobs University Bremen, JUB, Germany
KTH Royal Institute of Technology, KTH, Sweden
Oslo University College, HIO, Norway
Universitat Politècnica de Catalunya, UPC, Spain
University of Federal Armed Forces Munich, CETIM, Germany
Poznan Supercomputing and Networking Center, PSNC, Poland
University of Zürich, UniZH, Switzerland
Ludwig-Maximilian University Munich, LMU, Germany
University College London, UCL, UK
University of Pitesti, UniP, Romania

© Copyright 2008 the Members of the EMANICS Consortium

For more information on this document or the EMANICS Project, please contact:

Dr. Olivier Festor
Technopole de Nancy-Brabois - Campus scientifique
615, rue de Jardin Botanique - B.P. 101
F-54600 Villers Les Nancy Cedex
France
Phone: +33 383 59 30 66
Fax: +33 383 41 30 79
E-mail: <olivier.festor@loria.fr>

Document Control

Title: Open Source Support and Joint Software Development
Type: Public
Editor(s): Olivier Festor
E-mail: Olivier.Festor@loria.fr
Author(s): WP6 Partners
Doc ID: D6.4

AMENDMENT HISTORY

Version	Date	Author	Description/Comments
0.1	2008-11-20	J. Schönwälder	SMIng parser implementation
0.2	2008-02-16	F. Beck, O. Festor	NDPMon extensions
0.3	2008-02-22	D. Hausheer, M. Morariu	OpenDiameter packaging
0.4	2008-02-22	T. Bocek, F. Hecht, D. Peric	P2PFastSS implementation
0.5	2008-02-23	PSNC	Inventory data
0.6	2008-03-05	O. Festor	Integration, summary, introduction
0.7	2008-04-02	O. Festor	Quality check
0.8	2008-04-04	J. Schönwälder	ISMS implementation
0.9	2008-11-20	O. Festor	D6.4 evolution

Legal Notices

The information in this document is subject to change without notice.

The Members of the EMANICS Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the EMANICS Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Contents

1	Executive Summary	1
2	Introduction	2
3	SMIng Parser Implementation	4
3.1	Presentation	4
3.2	Libsmi Library	5
3.3	Implementation Progress Report	5
3.3.1	Data Structures	7
3.3.2	Functions	11
3.3.3	Smidump driver	13
3.4	Testing	14
3.5	Conclusions	14
4	NDPMon extensions	15
4.1	Presentation	15
4.2	Expected Impact	19
4.3	Progress Report	19
4.4	Conclusion	20
5	ISMS Implementation	23
5.1	Presentation	23
5.2	Expected Impact	24
5.3	Progress Report	25
5.3.1	TCP Nagle Interactions	25
5.3.2	SSH Window Adjustments	26
5.3.3	DTLS Retransmission Timers	26
5.3.4	Net-SNMP Limitations	26
5.4	Conclusions	27
6	P2PFastSS	28
6.1	Introduction	28
6.2	P2PFastSS Architecture	28
6.2.1	Indexing and Storing (First Phase)	29

6.2.2	Searching (Second Phase)	31
6.3	P2PFastSS Changes and Extensions	32
6.3.1	Disk-based Storage	33
6.3.2	Keyword Relevance	34
6.3.3	Paging of Search Results	35
6.3.4	Preview of Search Results	35
6.4	Impact Evaluation	36
7	NDPMon packaging and distribution	37
7.1	Presentation	37
7.2	Expected Impact	37
7.3	Progress Report	37
8	OpenDiameter Packaging and Distribution	40
8.1	OpenDiameter	40
8.2	Expected Impact	40
8.3	Progress Report	41
8.4	Debianizing Open Diameter	41
8.5	Debian Repository for Open Diameter Packages	42
9	An online catalogue of available software for network management	46
10	VoIPbot	52
10.1	Description	52
10.2	Progress report	54
10.3	Expected impact	57
11	LINUBIA	59
11.1	Presentation	59
11.2	Motivating Use Cases	60
11.2.1	Network Traffic Billing System	60
11.2.2	Individual Load Monitoring and Abuse Detection	60
11.2.3	Service Load Measurement	61
11.3	Design and Implementation	61
11.3.1	Requirements	61
11.3.2	LINUBIA Architecture	61

11.3.3 End-Host Accounting Architecture	62
11.3.4 Integrated View	63
11.3.5 Implementation	63
11.4 Evaluation	64
11.5 Expected Impact	66
11.6 Progress Report	66
11.7 Conclusion	66
12 NetViz	68
12.1 Presentation	68
12.2 Impact	69
12.3 Progress Report	69
12.3.1 set vsys <name> zone <number> vrouter <VR-name> id <id-number>	70
12.3.2 set vrouter <name> <command>	70
12.3.3 set zone <name> <command>	72
12.3.4 set interface <name> <command>	72
12.4 Conclusion	72
13 SMI to YANG Specification Translation	73
13.1 Presentation	73
13.1.1 Translation of Special Types	73
13.1.2 Module Prefix Generation Algorithm	74
13.1.3 Mapping of SMIv2 Modules	74
13.1.4 Mapping of SMIv2 Imports	75
13.1.5 Mapping of the MODULE-IDENTITY Macro	76
13.1.6 Mapping of the TEXTUAL-CONVENTION Macro	77
13.1.7 Mapping of the OBJECT-TYPE Macro	78
13.1.8 Mapping of the NOTIFICATION-TYPE Macro	82
13.2 Expected Impact	84
13.3 Progress Report	85
13.4 Conclusions	85

14 TomP2P: A Distributed Multi Map	87
14.1 Introduction	87
14.2 Distributed Hash Table	87
14.2.1 Fast Get	87
14.2.2 Reversed Conditional Get	88
14.3 Distributed Hash Table Extensions	88
14.3.1 Domains	88
14.3.2 Garbage Collection	89
14.3.3 Distributed Multi Map	89
14.4 Future Work	90
15 SNMPDump packaging and distribution	91
15.1 Presentation	91
15.2 Expected Impact	91
15.3 Progress Report	92
15.4 Conclusion	92
16 Weathermap packaging and distribution	93
16.1 Presentation	93
16.2 Expected Impact	93
16.3 Progress Report	93
16.4 Conclusion	93
17 Conclusions	94
18 Abbreviations	95
19 Acknowledgement	95

1 Executive Summary

Open Source support, coherent joint software development and packaging is a very important initiative for ensuring strong integration, network survivability and transfer of technology developed within the network. In the management community, many Open Source components have been produced over the years, some of them being widely used today for several purposes. Recent evolutions in management and networking technologies however did slow the use of these very valuable software components and the rhythm of new productions has decreased drastically. From the very beginning, EMANICS partners did recognize the need to support the Open Source initiatives in the area of management to foster their use and acceptance on large scale.

In June 2007, a first call has been issued to the EMANICS community containing two sub-calls : one for software development and one for Open Source software packaging. Four initiatives have been selected in the first call : SMIng support in LibSMI, P2PFastSS, ISMS implementation and VoIPbots. Two proposals have been selected in the packaging part: Open Diameter packaging and NDPMon distribution standardization and integration in standard distributions. In addition, the work on Management software categorisation, annotation and referencing did continue.

A second call was issued in april 2008. Like the first call of the second phase, it did contain two sub-calls : one on software development and one on software packaging. Five development projects were selected in this call (Lambda Monitor, Linubia, NetVis, SMIIYang, JYang) together with three packaging projects (TOMP2P, SNMPDump and Weathermap).

This report provides the details of the two calls (and four subcalls) that have been issued since June 2007 and for every selected project, a detailed description of the software package together with the achievements made so far. The work on the software inventory database is also reported in this document.

This report is an extension of deliverable D6.3 which was issued in december 2007. It closes the reporting of work-package 6 for the second phase of the EMANICS network of excellence.

2 Introduction

Open Source support, coherent joint software development and packaging is a very important initiative for ensuring strong integration, network survivability and transfer of technology developed within the network. In the management community, many Open Source components have been produced over the years, some of them being widely used today for several purposes. Recent evolutions in management and networking technologies however did slow the use of these very valuable software components and the rhythm of new productions has decreased drastically. From the very beginning, EMANICS partners did recognize the need to support the Open Source initiatives in the area of management to foster their use and acceptance on large scale.

This effort was already initiated in the first phase of the network of excellence. The results obtained there are reported in deliverable D6.2. In deliverable D6.3 we covered the activities undertaken from July 2007 to March 2008. This report entitled D6.4 is the revised version of this deliverable. It adds to the reporting all activities undertaken in this work package from April to December 2008.

In June 2007, two calls have been issued to the EMANICS community: one for software development and one for Open Source software packaging. Four initiatives have been selected in the first call : SMIng support in LibSMI, P2PFastSS, ISMS implementation and VoIPbots. All except one of these selected projects did start immediately after selection. Only the VoIPbot project was delayed at INRIA replaced by NDPMon extensions documented in this report. Two proposals have been selected in the packaging part of the first call : Open Diameter packaging and NDPMon distribution standardization and integration in standard distributions. In addition, the work on Management software categorisation, annotation and referencing did continue.

In April 2008, a second call was issued. Like the first call of the second phase, it did contain two sub-calls : one on software development and one on software packaging. The call received a total of eighteen proposals which is the highest number ever received since the start of the network. Out of these eighteen proposals, we selected five development projects (Lambda Monitor, Linubia, NetVis, SMiYang, JYang) together with three packaging projects (TOMP2P, SNMPDump and Weathermap). They are all reported in this deliverable.

To present the achievements, the report is structured as follows. Section 3 is dedicated to the LibSMI development for SMIng support, first selected development project in phase 2 of EMANICS. Section 4 contains the description of both the package and the new developments undertaken in phase 2 of EMANICS on NDPMon. Section 5 contains the ISMS development report. Section 6 presents the developments realized in the P2PFastSS similarity search funded initiative. Sections 7 and 8 are dedicated to the two selected packaging initiatives. Section 9 provides the description of enhancements brought to the software inventory and search service offered within EMANICS.

Extensions from D6.3 to D6.4 are reported starting from section 10 to the end of the report. The first software package reported is the VoIPbot package. Funded in part one of phase 2, its development was delayed to part 2. The second development supported under phase 2 is LambdaMonitor, reported in ???. Section 11 reports the development undertaken in the Linubia effort. Developments on visualization solutions using Juniper

management APIs are reported in 12. In the area of information model processing related to Netconf, two initiatives have been funded in this reporting period : SMMYang reported in section 13 and JYang a Java-based YANG processing engine reported in section ?? . The three supported packaging efforts are reported in section ??, 15 and 16.

Section 17 concludes this deliverable. The text of the two calls issued in July 2007 and April 2008 is given as an appendix to this document.

3 SMIng Parser Implementation

The `libsmi` library is an open source C library for parsing data models written in the SMI/SPPI languages. The library comes with a collection of tools to validate data models, to convert data models into various formats, and to analyze changes made between different versions of a data model for backwards compatibility. The library and the associated tools are widely using within the industry and for validation purposes by standardization bodies such as the IETF. The aim of this project was to add support for SMIng [1] to the `libsmi` library.

3.1 Presentation

The Structure of Management Information (SMI) is the language for describing Internet Management Information Bases (MIBs). There are two versions of the language that are in use: SMIv1 [2] and SMIv2 [3, 4, 5]. SMIv2 has superseded SMIv1 as the language used for MIB definitions, and most vendors use it even if they ship SMIv1 definitions to customers. This transition is supported by open source tools such as `libsmi`, which provides an API for accessing SMI structures and provides the tools for automatic conversion between the different SMI languages.

There are several problems with SMIv2. One of the main problems is its dependence on the ASN.1 syntax which is controlled by the ISO organization and not by the IETF. Furthermore the dependence is on an old version of ASN.1, which leads to many difficulties for developers who need more information on the syntactical rules of the [6]. The main difficulty is the implementation of parsers, which cannot rely on a BNF (Backus-Naur Form) grammar, which is the usual way for translation of language syntax to lexical and syntactical rules for parser generators. This leads to many developers using “fuzzy” MIB scanners that do not check for conformance with the SMI syntax [6]. Another problem with SMIv2 is that its syntax does not allow for efficient parsing, which causes many vendors to use more efficient but proprietary intermediate formats, which harms interoperability [6].

SMIng [1] is a new data definition language that addresses the problems of SMIv2. It is independent from other language definitions and is instead defined using ABNF grammar, which greatly simplifies the implementation of parsers [6]. Additionally SMIng syntax is greatly simplified because of a clear separation of statements by semicolon, which makes error recovery in parsers much easier [6]. It includes a more complete set of base data types. Its syntax is arguably simpler to read even for an experienced programmer that does not necessarily have experience with SMIng.

Further advantage of SMIng is its protocol independence. SMIng definitions are not tied to a specific network management protocol, so SMIng extensions, that define an ABNF grammar for conversion, are used for the actual mapping of SMIng definition to different concrete protocol implementations, such as SNMP [7] or COPS-PR [8] or NETCONF [9]. Another SMIng advantage are the language extensions, as that they will allow old parsers to work with newer definitions, should new syntax rules be added to the language [6].

The main statements in an SMIng module are: extensions, identities, type definitions (typedefs), and classes. Extensions provide mechanism for extending the language syntax, and to map its syntax to other languages. Identity statements define abstract and

untyped identities, and describe their semantics. They can be derived from another identity. Typedefs derive new data types by restricting the base types, or other derived types. The base types of SMIng are: 32 and 64-bit signed and unsigned integers; 32, 64, and 128-bit floating point numbers; enumeration type, that provides named numbers; named bits type for storing binary flags; object identifier type, used to describe the path in the registration tree for SMIv1/v2 nodes; and a pointer base type that is used to reference class or identity [1].

SMIng classes represent a container of related attributes and events. Attributes are either data types, or other class references. Events that represent asynchronous events related to the event statement's class. Classes can be derived from another class [1]

Despite its advantages, SMIng is not yet widely used. One reason is likely the lack of convenient programming tools to work with SMIng definitions. In particular, one of the obstacles to adoption of SMIng is the lack of an SMIng parser that provides a convenient API to access SMIng structures.

A potential tool for that task is `libsmi`, which is a portable C library that currently provides such an API for access to SMIv1 and SMIv2 definitions. This library hides parsing details providing accessing SMI definitions for applications that among many useful tasks can be used to translate between the different SMI versions.

The purpose of this project was to update `libsmi` to support the SMIng language. This included providing a lexical analyzer and parser for the language, updating the API to support iterators for the new features of SMIng and providing a dump tool that can be used to test the implementation, and to save module information from memory to the usual textual form of SMIng module.

The main goal of this library update is to create a useful tool on which future SMIng tools can be build and by that to promote adoption of SMIng.

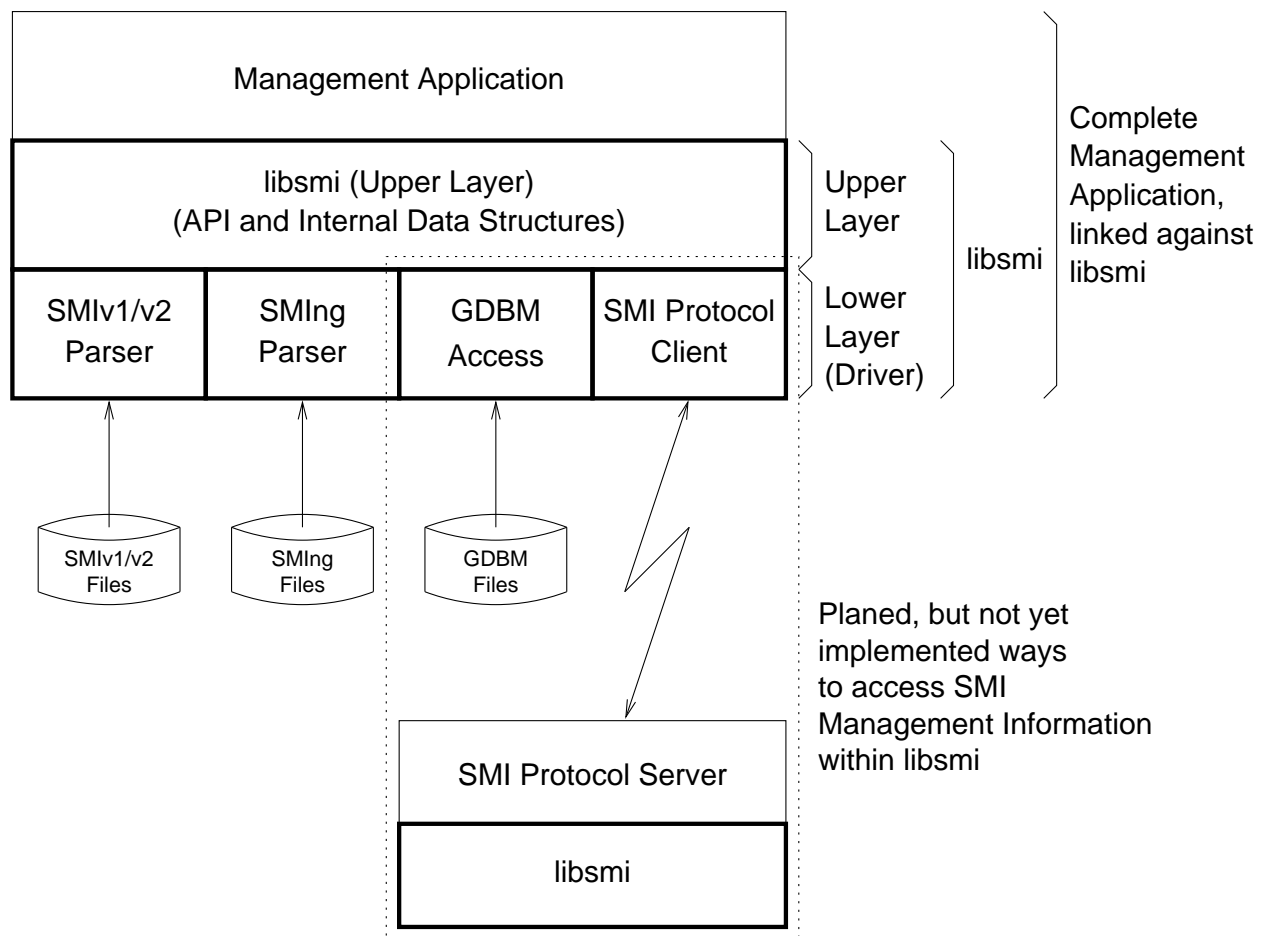
3.2 Libsmi Library

The `libsmi` library provides an API for accessing SMI module structures like modules definitions, type definitions, registered nodes in a registration tree, etc. The access can be done by iterators and by direct access to names or object identifiers. The structure of the library is two-layered (see Figure 1). The upper layer provides data structures to management information in memory and implements the API. The lower layer or the backends are the actual parsers that retrieve the management information from the SMI definitions. There are three currently available language back-ends for SMIv1 and SMIv2 and SPPI. There was an incomplete backend based on early version of SMIng, which was very different from the current language definition.

The goal of the project was to add support for the SMIng language as defined in [1], and to extend the `smidump` tool that comes with the library to dump SMIng modules.

3.3 Implementation Progress Report

The execution of the project started by implementing a lexical analyzer in `flex`. Apart from tokenizing keywords, identifiers, text segments, and number definitions, the `flex` scanner

Figure 1: Internal structure of the `libsmi` library

deals with removing comments and indentation from text segments, and checking of length constraints for identifiers.

The next level of parsing is done by a Gnu `bison` parser. It implements the `SMIng` ABNF grammar, with minor additions, to make some of the rules better suited for LR parsing. A major difference with the definition of `SMIng` in [1] with the implementation is the change of the placement of identity statements. Originally they were supposed to be placed after type definitions and before class definitions. The implementation changed identity statements to be before type definitions. The motivation behind this decision was to remove the forward references, which is one of the goals of `SMIng`. The problem with forward referencing came up because Pointer values in type definitions reference identities, so identities should be defined before the type definitions that reference them.

The first element of the `SMIng` definition that is read is the module and its import statements. If import statements are present, the corresponding modules are loaded into memory. For any statement that uses identifiers, a corresponding find function (like `findType()`, `findIdentity()`, `findClass()`, etc.) is used to check the identifiers for correct usage, that is to check whether such identifier is previously defined in case of a reference, or to check for repetition of names. If the referenced identifier is not found, or if it is already used for the same type of statement when defining new statement, the parser prints an error. The implementation, first searches in the current module for the corresponding statement

```
typedef struct SmiType {
    SmiIdentifier    name;
    SmiBasetype      basetype;
    SmiDecl          decl;
    char             *format;
    SmiValue          value;
    char             *units;
    SmiStatus         status;
    char             *description;
    char             *reference;
} SmiType;
```

Figure 2: External representation of a type in libsmi

```
typedef struct Type {
    SmiType          export;
    Module            *modulePtr;
    struct Type       *parentPtr;
    struct List       *listPtr;
    TypeFlags         flags;
    struct Type       *nextPtr;
    struct Type       *prevPtr;
    int               line;
} Type;
```

Figure 3: Internal representation of a type in libsmi

definition, then searches among the imported identifiers, and if an imported identifier with the same name is found, the corresponding module is searched for the corresponding statement definition.

3.3.1 Data Structures

The output of the parser is stored in memory in data structures internal to libsmi. The data structures also have two layers: public and private. Most of the API functions return structures that contain the data of a given statement and are named with an smi prefix, for example SmiType, SmiIdentity, SmiModule, etc. These structures contain the data for the given statements and include the description, status, reference, and other data fields specific for the type of statement. For example the SmiType structure contains a field for a default value. The private part of the data structures holds lists of the corresponding smi structures, and some data that is meant to be accessed through functions, such as parent types, modules associated with the current statement and others. The private structures all are named after their smi-prefixed counterparts. A representative example is SmiType shown in Figure 2 and the private structure Type shown in Figure 3, which contains a member struct SmiType export.

```
typedef struct SmiIdentity {
    SmiIdentifier    name;
    SmiDecl          decl;
    SmiStatus        status;
    char             *description;
    char             *reference;
} SmiIdentity;
```

Figure 4: External representation of an SMIng identity in libsmi

```
typedef struct SmiClass {
    SmiIdentifier    name;
    SmiDecl          decl;
    SmiStatus        status;
    char             *description;
    char             *reference;
} SmiClass;
```

Figure 5: External representation of an SMIng class in libsmi

As SMIv1, SMIv2, and SMIng are used to describe similar data, there are some similarities in the logic behind the languages which leads to the possibility of reuse of the internal data structures, and to transcode SMIv2 modules to SMIng and vice-versa, at least for modules that contain only type and extension definitions. However, for full conversion of SMI modules, one needs SMIng extensions to define the complete SMIng mapping to SMIv2 [10]. The conversion between SMI languages was beyond the scope of this project.

The data structures that were reused were the `SmiType`, `SmiModule`, `SmiImport`, and `SmiMacro`. The `SmiModule`, `SmiType`, and `SmiImport` structures were unchanged from the last version of the library. The `SmiMacro` struct, which holds SMIng extensions, was extended with an `abnf` string member, to hold the ABNF information of SMIng extension. The `SmiValue` value union, which holds default values for `SmiType`, was updated with the floating point number types.

The public structures contain fields of enumerated C types, which define the different values this fields can hold. Several of this enumerated types were extended with new definitions specific to SMIng. The `SmiBasetype` was extended with floating point values `SMI_BASETTYPE_FLOAT32`, `SMI_BASETTYPE_FLOAT64`, `SMI_BASETTYPE_FLOAT128`, and the new SMIng pointer base type `SMI_BASETTYPE_POINTER`. The `SmiAccess` enum was extended with the SMIng `SMI_ACCESS_EVENT_ONLY` access mode. `SMI_DECL_IDENTITY`, `SMI_DECL_CLASS`, `SMI_DECL_ATTRIBUTE`, and `SMI_DECL_EVENT` were added to the `SmiDecl` enumeration that defines type of statements and can be used to retrieve definitions.

The new public structures that were added were `SmiEvent` representing class events (Figure 7), `SmiAttribute` representing class attributes (Figure 6), `SmiClass` representing classes (Figure 5) and `SmiIdentity` for SMIng identity definitions (Figure 4). Several private structures were added corresponding to the new public structures as shown in Figures 8, 9, 10, and 11.

```
typedef struct SmiAttribute {
    SmiIdentifier    name;
    SmiBasetype      basetype;
    SmiDecl          decl;
    char             *format;
    SmiValue          value;
    char             *units;
    SmiStatus         status;
    char             *description;
    char             *reference;
    SmiAccess         access;
} SmiAttribute;
```

Figure 6: External representation of an SMIng attribute in libsmi

```
typedef struct SmiEvent {
    SmiIdentifier    name;
    SmiDecl          decl;
    SmiStatus         status;
    char             *description;
    char             *reference;
} SmiEvent;
```

Figure 7: External representation of an SMIng event in libsmi

```
typedef struct Identity {
    SmiIdentity       export;
    Module            *modulePtr;
    struct Identity   *parentPtr;
    struct Identity   *nextPtr;
    struct Identity   *prevPtr;
    int               line;
} Identity;
```

Figure 8: Internal representation of an SMIng identity in libsmi

```
typedef struct Class {
    SmiClass          export;
    Module            *modulePtr;
    struct Attribute  *firstAttributePtr;
    struct Attribute  *lastAttributePtr;
    struct List        *uniqueList;
    struct Event       *firstEventPtr;
    struct Event       *lastEventPtr;
    struct Class       *parentPtr;
    struct Class       *nextPtr;
    struct Class       *prevPtr;
    int                line;
} Class;
```

Figure 9: Internal representation of an SMIng class in libsmi

```
typedef struct Attribute {
    SmiAttribute      export;
    Class              *classPtr;
    struct Type        *parentTypePtr;
    struct List        *listPtr;
    struct Attribute   *nextPtr;
    struct Attribute   *prevPtr;
    int                line;
    struct Class       *parentClassPtr;
} Attribute;
```

Figure 10: Internal representation of an SMIng attribute in libsmi

```
typedef struct Event {
    SmiEvent          export;
    Class              *classPtr;
    struct Event       *nextPtr;
    struct Event       *prevPtr;
    int                line;
} Event;
```

Figure 11: Internal representation of an SMIng event in libsmi

All the private structures are elements of doubly-linked lists and have references to the previous and the next element. All of them have as first element a corresponding `libsmi` public structure. This is used by internal library functions to cast public structure pointers to private structure pointers. The `Class` and `Identity` structures have pointers to their module and to the parent class or identity from which they are derived. The `Attribute` and `Event` structures do not have a module pointer, but instead have a `Class` pointer, as they are tied to their class and not directly to a module. Each `Attribute` has either a parent class or parent type specified through the corresponding pointer. The `Class` has pointers to a list of the `Attribute` structures that it contains and to a list of unique elements. The list is either empty (`uniqueList=NULL`) if the class is not meant to be instantiated separately, or the list's first element points to the current class (`Class->uniqueList.ptr = Class`) if the class is scalar, that is its unique list is empty and it is meant to have only one instance, or contains a list of `Attribute` structures, that uniquely defines the class instance.

3.3.2 Functions

The private API of the library was extended with setter functions like `setClassName()`, `setClassDecl()`, `addClass()` and other similar methods for the `Identity`, `Event`, and `Attribute` structures. The `Event` and `Attribute` functions differ from `Identity` and `Class` functions in the fact that for example `addClass()` function takes as one of its arguments `Module` pointer, whereas `Event` and `Attribute` functions take as argument, a `Class` pointer, because they are tied to their class, and not directly to the module. The `Attribute` structure also has a `duplicateTypeToAttribute()` function, that duplicates an existing `Type` to an `Attribute`.

The public API was extended with the corresponding `smiGet*()` iterators, for the `Identity`, `Class`, `Event`, and `Attribute` structures:

- The function `smiGetFirstIdentity(SmiModule *smiModulePtr)` together with the function `smiGetNextIdentity(SmiIdentity *smiIdentityPtr)` are used to iterate through the identities of the module given by `smiModulePtr`. They return a pointer to `struct SmiIdentity` that represents an identity or `NULL` if there are no identities left in the module, or an error has occurred.
- The function `smiGetIdentity(SmiModule *smiModulePtr, char *name)` returns a pointer to the `struct SmiIdentity` for the identity with the given name in the given module, or `NULL` if the identity with the given name does not exist.
- The function `smiGetIdentityModule(SmiIdentity *smiIdentityPtr)` returns a pointer to `struct SmiModule` of the module containing the given identity.
- The function `smiGetParentIdentity(SmiIdentity *smiIdentityPtr)` returns a pointer to a `struct SmiIdentity` pointing to the parent of the given `smiIdentityPtr`, or `NULL` if the identity is not derived.
- The function `smiGetFirstClass(SmiModule *smiModulePtr)` together with the function `smiGetNextClass(SmiClass *smiClassPtr)` are used to iterate through all the classes of the module given by `smiModulePtr`. They return a pointer to `struct`

`SmiClass` that represents a class or NULL if there are no classes left in the module, or an error has occurred.

- The function `smiGetClass(SmiModule *smiModulePtr, char *name)` returns a pointer to struct `SmiClass` that represents the class with the given name in the current module, or NULL if the class with the given name does not exist.
- The function `smiGetClassModule(SmiClass *smiClassPtr)` returns a pointer to struct `SmiModule` of the module containing the given class.
- The function `smiGetParentClass(SmiClass *smiClassPtr)` returns a pointer to struct `SmiClass` pointing to the parent of the given `smiClassPtr`, or NULL if the class is not derived.
- The function `smiIsClassScalar(SmiClass *smiClassPtr)` returns int 1 if the class is a scalar (its unique statement contains an empty list) or 0 otherwise. This function can be used in conjunction with `smiGetFirstUniqueAttribute()` to determine whether the class is meant to be instantiated separately (has unique statement with nonempty list), or if it is meant to be used as part of another class (has no unique statement).

Some of the fields of an `SmiAttribute` may be empty if it references a class instead of a type. There are `getAttributeParent*()` functions for retrieving attribute parent, type or class. They can be used to determine if an attribute references a type or class. Also functions for retrieving named numbers and ranges of `SmiAttribute`'s, were added.

- The function `smiGetAttribute(SmiClass *smiClassPtr, char *name)` returns a pointer to the struct `SmiAttribute` for the attribute with the given name in the current module, or NULL if the attribute with the given name does not exist.
- The function `smiGetFirstAttribute(SmiClass *smiClassPtr)` together with the function `smiGetNextAttribute(SmiAttribute *smiAttributePtr)` are used to iterate through the attributes of the class given by `smiClassPtr`. They return a pointer to a struct `SmiAttribute` that represents an attribute or NULL, if there are no attributes left in the class, or an error has occurred.
- The function `smiGetAttributeParentClass(SmiAttribute *smiAttributePtr)` returns pointer to a struct `SmiClass` pointing to the parent of the attribute identified by `smiAttributePtr`, or NULL if the attribute does not reference class. Note that attributes always have either parent type or parent class.
- The function `smiGetAttributeParentType(SmiType *smiAttributePtr)` returns a pointer to a struct `SmiType` pointing to the parent type of the given `smiAttributePtr`, or NULL if the attribute does not reference type. Note that attributes always have either parent type or parent class. The functions `smiGetAttributeParentClass()` and `smiGetAttributeParentType()` can be used to determine what type reference, class or type, the given attribute is.

- The function `smiGetFirstUniqueAttribute(SmiClass *smiClassPtr)` together with the function `smiGetNextUniqueAttribute(SmiType *smiAttributePtr)` are used to iterate through the unique attributes of the module given by `smiClassPtr`. They return a pointer to struct `SmiAttribute` that represents a unique attribute or `NULL` if there are no unique attributes left in the class, or an error has occurred. This function **MUST NOT** be used for scalar classes, so it should only be called after `isClassScalar()` has returned 0.
- The function `smiGetEvent(SmiClass *smiClassPtr, char *name)` returns a pointer to the struct `SmiEvent` for the attribute with the given name in the current module, or `NULL` if the event with the given name does not exist.
- The function `smiGetFirstEvent(SmiClass *smiClassPtr)` together with the function `smiGetNextEvent(SmiEvent *smiEventPtr)` are used to iterate through the events of the class given by `smiClassPtr`. They return a pointer to struct `SmiEvent` that represents an event or `NULL` if there are no events left in the class, or an error has occurred.
- The functions `smiGetAttributeFirstRange(SmiAttribute *smiAttributePtr)` and `smiGetAttributeNextRange(SmiRange *smiRangePtr)` are used to iterate through ranges that restrict number or octet string types. Both functions return a pointer to the struct `SmiRange` representing the range, or `NULL` if there are no more ranges, or an error has occurred.
- The functions `smiGetAttributeFirstNamedNumber(SmiAttribute *smiAttributePtr)` and `smiGetAttributeNextNamedNumber(SmiNamedNumber *smiNamedNumberPtr)` are used to iterate through named numbers of bits or enumerations for attributes, which reference types, and to retrieve the reference restriction of a pointer. Both functions return a pointer to the struct `SmiNamedNumber` representing the named number, or `NULL` if there are no named numbers left, or an error has occurred. The function `smiGetFirstNamedNumber()` can be used to retrieve the name of the identity that is restricting a pointer type, as it is stored as the name of the first named number.

3.3.3 Smidump driver

The final part of the project was to update the dump tool `smidump`, which is part of the `libsmi` package. The `smidump` tool has several drivers for dumping SMI structures into different formats. As part of the EMANICS funded development, the SMIng driver was updated to print the memory structures of `libsmi` back to text format according to the SMIng module definitions. The tool's most important task within the project was that it was used for testing of the parser. The driver prints the data structures information and the corresponding keywords, that would form a valid module, semantically identical to the the input module.

3.4 Testing

Initial testing was done manually by writing short modules that contains single, or in case of referencing tests several, statements, and checking whether the retrieval functions (the `smiGet*()` functions), return structures with the same information as the input. This was done to decouple the testing of the parser from the testing of `smidump`.

Several modules were written, to represent most of the statement combinations that occur in SMIng. More specifically typedefs for all the base types were written, and checked whether the internally stored information matches the initial module definition. Additionally derived types were generated to check whether derivation of types works correctly. The same was done for identities, extensions, and classes. Classes' attribute and event statements were also tested for correctness.

Consistency tests were performed for all types of statements that are defined by referencing. They were checked with modules that have incorrect references (for example a typedef that derives from not yet defined type, or an identity that is derived from an undefined or non-imported parent identity). Tests with modules that have redefinition of identifiers were performed as well. The parser raises errors in all cases where inconsistent reference or redefined name exists.

After the parser was checked component-wise, an SMIng driver was written for `smidump` and several larger modules comprising of combination of the smaller test cases were tested with `smidump`. The output of the dump tool was manually checked with the help of `diff` for differences between the output module and the original module, and no semantic differences were found. Some other modules that were used for testing the parser were the IRTF-NMRG-SMING modules that are distributed as part of `libsmi` MIB module collection.

3.5 Conclusions

Although the parser works, as expected, no real-world users have tested it so far, which means that there might be yet unknown bugs, or features perceived as bugs. In order to ensure the quality of the new functionality of `libsmi` added by this project, the SMIng parser will need to gain some user base. Although the API includes all the essential functions for retrieving information from SMIng modules, some usage patterns might not be known at the moment, and future feature requests to address user needs are possible.

The SMIng language has not yet gained industrial application, but its advantages are numerous and recognized in the management community. The `libsmi` library, with its SMIng parser, is a very useful tool, providing an easy way to write programs that use SMIng definitions. However, there is room for future improvement of the library. The first important and needed improvement is the update of the drivers of `smidump` adding SMIng support. Another important improvement is the creation of a tool that can convert SMlv1/v2 modules to SMIng modules, and vice versa, which is essential for transition from older MIB implementations and compatibility with them. This feature might be integrated into dump drivers, as a separate application, or as a feature of the `libsmi` library itself. Despite the needed improvements, `libsmi` already provides good base for creation of SMIng capable applications.

4 NDPMon extensions

4.1 Presentation

NDPMon is Open Source software for IPv6 Neighbor Discovery Protocol monitoring. It is distributed under the LGPL licence as a SourceForge at the URL <http://ndpmon.sf.net>. Initially designed with a grant from CISCO Systems in 2006, the development of the tool was pursued with EMANCIS funding in 2007 where it first went public. Since a couple of extensions and portings have been realized. They are reported in this section.

Overview

ArpWatch (<http://ee.lbl.gov>) is used in many networks and is well known by network administrators as one of the core components of the monitoring plane for IPv4 networks. Its role is to monitor the activity of the ARP protocol. By analyzing ARP packets, it maintains an up-to-date cache in which the pairing between IPv4 and Ethernet addresses for all hosts on the link are stored. A timestamp is associated to each entry in this cache, which enables a monitoring over the time of host activity. When an ARP packet is captured, it is compared to the information in the cache, and if a suspicious behavior, or special activities (a new station has appeared in the network) are detected, a report is sent to the administrator. These reports are sent via various channels including syslog and mail for the most common ones.

NDPMon, Neighbor Discovery Protocol Monitor, is an IPv6 version of ArpWatch. It initially performs the same monitoring tasks as ArpWatch, but for IPv6. It is in charge of monitoring the Neighbor Discovery Protocol activities and maintains a neighbor database up-to-date, which contains the correspondences between IPv6 and Ethernet addresses, alongside with a Time-stamp. When a Neighbor Discovery packet is captured, the content is compared to the entries in the database. Usually, in IPv4, only one address was assigned to an interface. In IPv6, multihoming is one of the key features, for Network Renumbering [?] for example. When defining the neighbors cache entries, we have to take this specificity into account. In the same way than ArpWatch, activities and suspicious behaviors raise alerts and reports. All these alerts are syslogged, and depending on the syslog daemon configuration, they can be sent to a remote station. The most severe ones, namely the suspicious activities, raise alerts, which are sent by mail to a defined address, by using an external Mail Transfer Agent (MTA).

In addition to this monitoring role, NDPMon is also able to detect attacks against the Neighbor Discovery Protocol, as defined in [?], misconfiguration, stack vulnerabilities and suspicious behaviors.

Architecture

The tool is designed to be deployed on each subnet or link of the network, as shown in figure 12. It operates in two phases: a learning phase and a monitoring phase. During

the learning phase, NDPMon builds the neighbors database by capturing the Neighbor Discovery messages. During this phase, the tool does not send any report. NDPMon makes the assumption that, when it enters the learning phase, the network is healthy and that all the activities are legitimate.

Once this phase is over, the tool can switch to monitoring mode. Thus, based on the Router Advertisements received, it also populates the routers list. Switching can be done either remotely or by restarting the daemon with the right parameters.

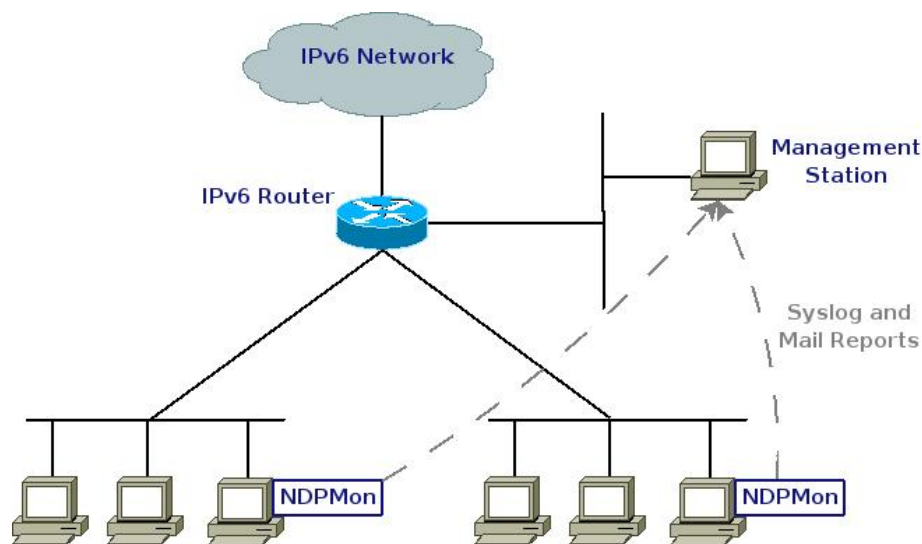


Figure 12: NDPMon Deployment

The software that implements NDPMon has been coded in the C language. It must be launched with root privileges and runs as a daemon. To limit as much as possible the resource consumption, we performed some code profiling with the tool ValGrind (<http://valgrind.org/>). NDPMon uses two XML files, which are parsed and modified by libxml2 (<http://xmlsoft.org/>) (see Figure 13).

The first file is the configuration file, which contains two main parts. First, there are some options for the configuration of the daemon itself: syslog facility to use, Email address for the reports... The administrator must also give the list of the legitimate routers on the link (Ethernet and IPv6 addresses, advertised IPv6 prefixes). This information is used by the daemon as the legitimate routing data on the link, and is used as references when Neighbor Discovery packets are received. If the tool is launched in learning phase, the information related to the authorized routers is completed with the information contained in the received Router Advertisements.

The second file is the neighbor cache. It contains the pairings between IPv6 and Ethernet addresses. A time-stamp is set for each entry in the cache. This file is filled automatically by the daemon. If NDPMon runs in learning phase, this database will be constructed without sending any report.

When the tool is launched, these two files are parsed. The neighbor cache is loaded and the entries are converted in C structures. The routing information in the configuration file is converted in the same way. The daemon begins to listen to the network and captures

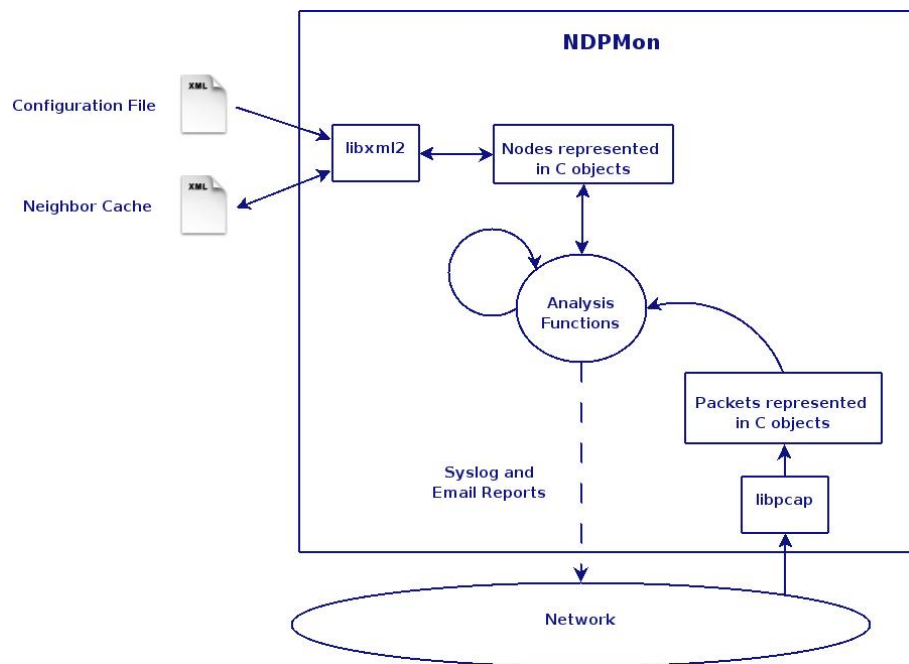


Figure 13: NDPMon Implementation Architecture

Neighbor Discovery packets thanks to the libpcap (<http://www.tcpdump.org/>). These packets are converted in C structures. The tool implements analysis functions to verify if the packet is legitimate, or if it is a malicious packet.

If the packet is considered malicious, reports are sent via syslog, and, depending on the severity, a mail is sent to the administrator. If the packet is legitimate, the information it contains is added in the neighbor cache if they are not already present; otherwise the Time-stamp of the corresponding is simply updated.

Monitored Activities

The Neighbor Discovery activities monitored by the tool are:

- New station: a new Ethernet address appears on the network, a new node has appeared
- New IPv6 Global Address: a new IPv6 global address is detected for a host
- New Link Local Address: a new IPv6 link local address is detected for a host
- New activity: the source node had no activity during the last month
- Bogon: the IPv6 source address is not local to the link
- Ethernet mismatch: the Ethernet address specified in the ICMP option is not the same than the Ethernet source address of the packet

- Changed ethernet address: a node changed its Ethernet address while keeping the same IPv6 address
- Flip flop: a node is switching between two different Ethernet addresses
- Reused address: a node is re-using an old Ethernet address

There are several possible attacks against the Neighbor Discovery Protocol. Monitoring only the classic Neighbor Discovery activities is not sufficient for detecting them. To detect these attacks we must, besides the neighbors cache, also define the legitimate behavior for the routing on the link. By defining the router Ethernet and IPv6 valid addresses, and the legitimate prefixes on the link, we can detect the following attacks:

- Wrong MAC/IP pair: Separately, the MAC and IP addresses are valid, but not as a couple
- Unknown MAC Manufacturer: the vendors of the MAC address is not known, it may be forged
- Wrong router mac: the Ethernet source address of the Router Advertisement is not defined as valid
- Wrong router ip: the IPv6 source address of the Router Advertisement is not defined as valid
- Wrong prefix: the prefix advertised in the Router Advertisement is not legitimate on the link
- Wrong router redirect: the source of the Redirect message is not a legitimate router
- NA router flag: the Neighbor Advertisement has the Router flag set whereas it is not defined in the known routers list
- DAD DoS: Denial of Service toward the Duplicate Address Detection mechanism
- Ethernet broadcast: the Ethernet source address is the broadcast address
- IP multicast: the IPv6 source address is a specific multicast address

When detected, all these behaviors trigger the sending of syslog and mails alerts. Some of them are not only the manifestation of a malicious behavior or a misconfiguration; they symbolize vulnerabilities in the IPv6 stack. For example, depending on the version of the Linux kernel, and thus of the IPv6 stack, it is possible or not to assign an Ethernet broadcast address on an interface. If such a behavior is detected, it means that the host on which this address is set has an old version of the IPv6 stack, and that it could be judicious to update it.

4.2 Expected Impact

The software is meant to be deployed in all IPv6 networks to help administrators to monitor the nodes activity, detect unexpected behaviors or presence of nodes on their network, and collect some statistics about it. We would like to deploy it on as many networks as possible, to gather as much feedback and experiences as possible.

4.3 Progress Report

In this section, we will present the progresses and new features brought to NDPMon since the beginning of the EMANICS support campaign in 2007.

New Alerts

By taking as references the RFC2461 (IPv6 Neighbor Discovery Protocol [?]) and RFC2462 (IPv6 Stateless Address Autoconfiguration [?]), we identified new threats and implemented the associated attacks:

- Wrong ipv6 router: if neither the Link Local Address and the MAC address are known for a RA
- Wrong RA flags: if the managed and other flags in the RA are not well set
- Wrong source link address option: the MAC address in the Link Address option does not match with the Ethernet source address
- Wrong ipv6 hop limit: IPv6 Hop Limit is not 255
- Wrong RA lifetimes: preferred lifetime is bigger than the valid lifetime
- RA valid lifetime too short: valid lifetime is less than 2 hours

These new alerts are integrated in version 1.3c dated from the 25 of October 2007.

Multi-OS Porting

NDPMon has been ported under the following new Operating Systems:

- FreeBSD - release 1.2 31 July 2007
- OpenBSD release 1.2a - 28 August 2007
- NetBSD release 1.3 - 10 September 2007
- Mac OS X 10.4 Tiger release 1.3 - 10 September 2007

GUI

Alongside with release 1.3a from the 25 of September 2007, we implemented and released a User Interface for NDPMon. It is actually a WEB interface. The daemon has been modified as follows:

- All alerts are piped through the Python script `create_html_table.py` into an XML file `alerts.xml`
- Each time the neighbors cache is written, the daemon stores the number of neighbors currently discovered with a timestamp in the file `discovery_history.dat`

These two files are used by the WEB interface, the first one to display the alerts and reports in an HTML table with colours codes (green is for information, orange for warning, and red for problem), as shown in figure 14, and the second one to generate statistics over the discovery.

Time	Reason	MAC Address	Vendor	IPv6 Address
Tue Sep 18 12:01:59 2007	new station	0:c:f1:82:31:43	Intel	fe80:0:0:20c:f1:82:31:43
Tue Sep 18 12:01:31 2007	bogon	0:30:b6:51:04:1c	Cisco	2001:660:4501:32:0:0:0:1
Tue Sep 18 12:00:11 2007	wrong couple IP/MAC	0:13:72:35:7:45	Dell	fe80:0:0:230:b0fffe51:041c
Tue Sep 18 11:56:18 2007	new station	0:f1:b6:51:04:1c	WvPcbATest	fe80:0:0:e90e:a10c:1baa:bc05
Tue Sep 18 11:56:17 2007	wrong prefix	0:30:b6:51:04:1c	Cisco	fe80:0:0:230:b0fffe51:041c
Tue Sep 18 11:56:17 2007	new station	0:18:0b:ad:1167	Dell	fe80:0:0:ec42:7883:748a:651b
Tue Sep 18 11:56:17 2007	new station	0:16:0c:59:34:5b	AsustekCom	fe80:0:0:a5f7:befc:75d:3a6
Tue Sep 18 11:56:17 2007	new station	0:16:04:ae:62:a6	CompaqComm	fe80:0:0:3a76:e0d5:99a2:6a66
Tue Sep 18 11:56:09 2007	new station	0:2:a5:63:1a:66	CompaqComp	fe80:0:0:202:a5fffe63:1a66
Tue Sep 18 11:55:57 2007	new lla	0:30:b6:51:04:1c	Cisco	fe80:0:0:230:b0fffe51:041c
Tue Sep 18 11:55:53 2007	new lla	0:30:b6:51:04:1c	Cisco	fe80:0:0:202:a5fffe63:1a66
Tue Sep 18 11:55:52 2007	ethernet mismatch	0:2:a5:63:1a:66	CompaqComp	fe80:0:0:202:a5fffe63:1a66
Tue Sep 18 11:52:32 2007	new station	0:14:51:5:8a:8a	AppleCompu	fe80:0:0:214:51f605:8a8a
Tue Sep 18 11:48:48 2007	wrong router ip	0:30:b6:51:04:1c	Cisco	2001:660:4501:1:213:72fffe35:745
Tue Sep 18 11:47:26 2007	new lla	0:13:72:35:7:45	Dell	fe80:0:0:213:72fffe35:745
Tue Sep 18 11:47:22 2007	new station	0:18:0b:72:44:79	Dell	fe80:0:0:6155:f22a:82ba:c483
Tue Sep 18 11:46:38 2007	new station	0:f1:f0:0:ac:c6	WvPcbATest	fe80:0:0:201fffe8:bcc6
Tue Sep 18 11:45:42 2007	new station	0:d:93:77:ed:0	AppleCompu	fe80:0:0:20d:93fffe77:ed00
Tue Sep 18 11:43:49 2007	new station	0:13:72:35:7:45	Dell	2001:660:4501:1:213:72fffe35:745
Tue Sep 18 11:43:44 2007	bogon	0:30:b6:51:04:1c	Cisco	2001:660:4501:32:0:0:0:1
Tue Sep 18 11:41:51 2007	new station	0:14:22:36:b8:9e	Dell	fe80:0:0:214:22fffe36:b89e
Tue Sep 18 11:41:45 2007	tip flip	0:13:72:35:7:45	Dell	fe80:0:0:230:b0fffe51:041c
Tue Sep 18 11:41:44 2007	tip flip	0:30:b6:51:04:1c	Cisco	fe80:0:0:230:b0fffe51:041c
Tue Sep 18 11:41:40 2007	changed ethernet address	0:13:72:35:7:45	Dell	fe80:0:0:230:b0fffe51:041c
Tue Sep 18 11:41:40 2007	wrong router mac	0:13:72:35:7:45	Dell	fe80:0:0:230:b0fffe51:041c
Tue Sep 18 11:41:17 2007	new station	0:18:c5:c1:50:2f	Dell	fe80:0:0:215:c5fffec1:502f
Tue Sep 18 11:40:47 2007	new station	0:30:b6:51:04:1c	Cisco	fe80:0:0:230:b0fffe51:041c
Tue Sep 18 11:29:17 2007	new station	0:d:93:c5:1a:a4	AppleCompu	fe80:0:0:20d:93fffe51:a4a4

Figure 14: NDPMon WEB Interface - Alerts

All information contained in the XML files (configuration, alerts and neighbors) is displayed in HTML tables via XSL transformations. The WEB interface generates statistics over the discovered nodes and the Mac Manufacturers as shown in figure 15.

4.4 Conclusion

The successive support initiatives of EMANCIS have enabled the maturation of the NDPMon framework. The tool is now complete, tested and operational for large Ipv6 networks.

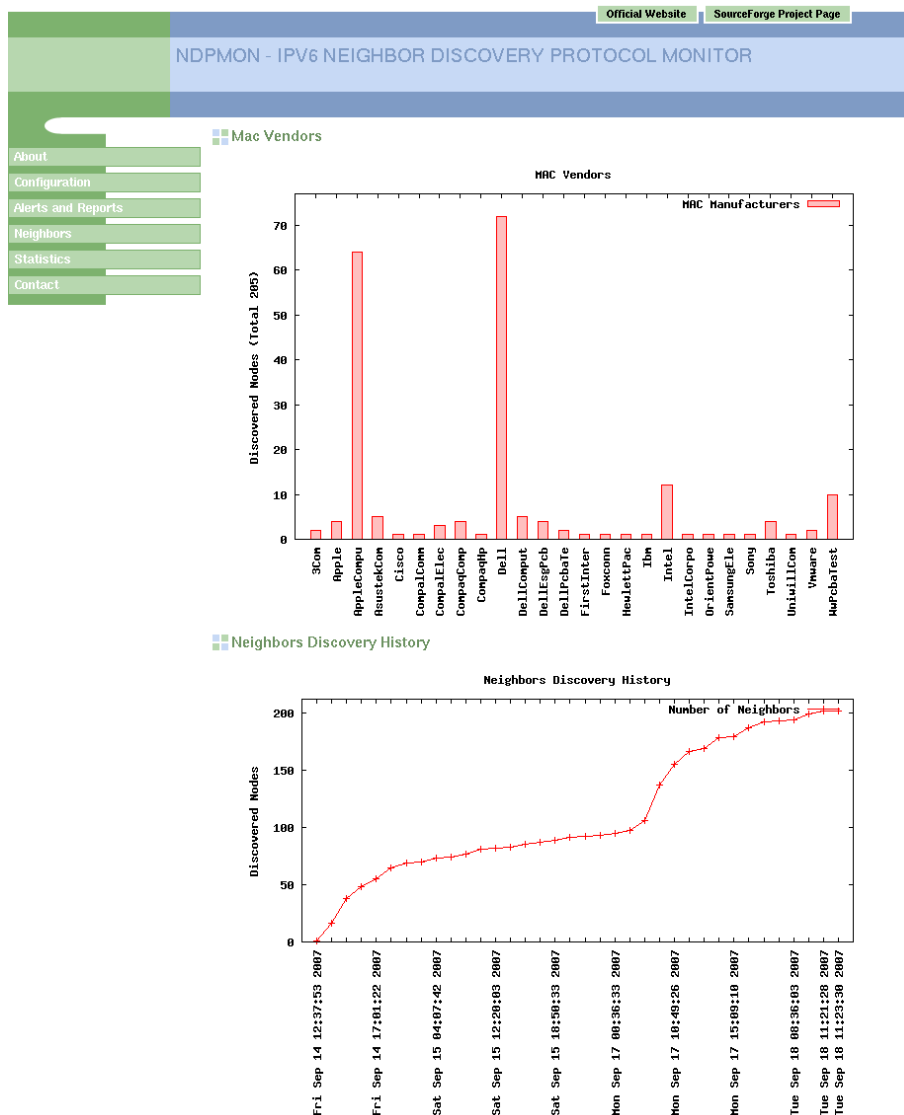


Figure 15: NDPMon WEB Interface - Statistics

Its full-fledged graphical user interface and multi-OS availability are strong elements for its acceptance. We currently witness an increase in the community around the framework and are aware of several large scale real deployments which is very encouraging.

5 ISMS Implementation

The Integrated Security Model for SNMP working group (ISMS) of the Internet Engineering Task Force (IETF) is working on specifications that extend the SNMP architecture [11] with a transport subsystem [12] and a transport security model [13] so that SNMP can run over secure transport protocols, such as SSH or TLS. The ISMS working group selected SSH [14] as a secure transport required to implement [15]. This project supports a prototype implementation of the ISMS specifications as part of the open source *Net-SNMP* package.

5.1 Presentation

The SNMP architecture defined in [11] and extended in [12] is designed to be modular in order to support future protocol extensions such as additional security models or additional transports. The architecture defines several subsystems and interfaces between these subsystems that should remain unchanged when subsystems are extended. The goal was to reduce side effects that can occur without such an architectural framework when the protocol is extended.

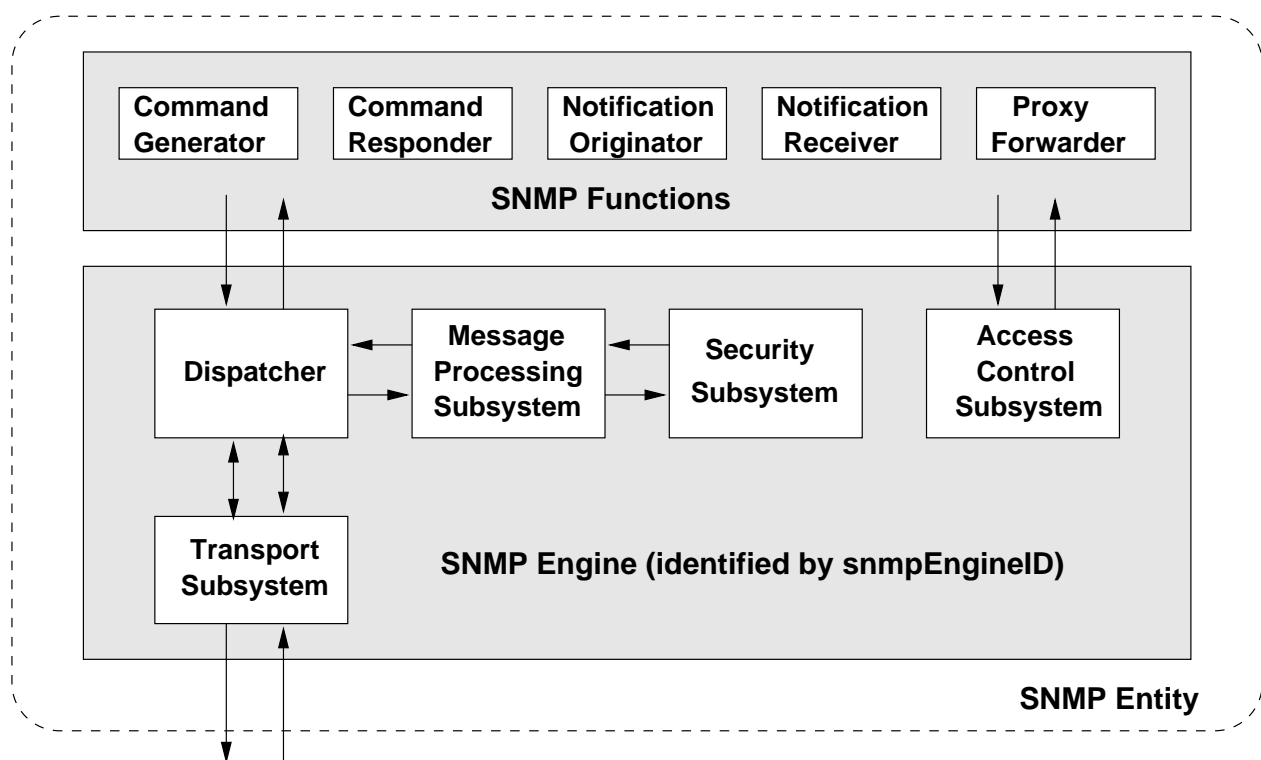


Figure 16: Structure of an SNMP entity according to the SNMPv3 architecture

According to the SNMP architecture defined in [11] and extended in [12], an SNMP engine consists of a message processing subsystem, a security subsystem, an access control subsystem, a transport subsystem and a single dispatcher (Fig. 16). Each subsystem can contain multiple concrete models that implement the services provided by that subsystem. The interfaces between subsystems are defined as Abstract Service Interfaces

(ASIs). The dispatcher is a special component controlling the data flow from the underlying transports through the SNMP engine and up to the SNMP applications. The dispatcher is a singleton and a fixed part of the architecture.

Most existing SNMPv3 implementations support three message processing models for SNMPv1, SNMPv2c, and SNMPv3 and two security models, namely the User-based Security Model (USM) [16] (used by the SNMPv3 message processing model) and the Community-based Security Model (CSM) [17] (used by the SNMPv1 and SNMPv2c message processing models). There is only a single View-based Access Control Model (VACM) so far. In SNMPv1, SNMPv2c and SNMPv3/USM, security services (authentication, data integrity checking, encryption) are provided by the security subsystems, which is called from the message processing subsystem. This approach is called *message-based security* since all security processing happens on a per SNMP message basis. In particular, there is no notion of a security session and hence there is also no notion of session keys.

The transport subsystem [12], which has been recently defined by the ISMS working group of the IETF and was not part of the original SNMP architecture, permits multiple transport protocols to be "plugged into" an SNMP engine. The transport protocols are represented by transport models and may be security-aware.

Particularly interesting are secure transport protocols such as SSH, TLS, or DTLS that are wrapped by corresponding transport models, namely the SSH Transport Model (SSTHM) [15] or the TLS Transport Model (TLSTM) [18]. These secure transport models have in common that they have a notion of a session and provide security services (authentication, data integrity checking, encryption) on a per session basis, so called *session-based security*. Furthermore, the secure transport protocols all have a notion of a session key, which is generated out of initial keying material and valid only for the duration of one session.

The SNMP architecture was not designed with session-based security in mind. As a consequence, the original ASIs between the subsystems do not pass all the necessary security information to all subsystems. In order to minimize the changes to the original architecture, a new Transport Security Model (TSM) [13] for the security subsystem was introduced. The TSM sits where traditionally message-based security services are provided and it interacts with session-based secure transports through a shared cache. The necessary translation of transport-specific security parameters (e.g., the name of the authenticated SSH user) and the SNMP-specific, model-independent parameters (e.g., `securityName` and `securityLevel`) happens within a secure transport model (e.g., the SSTHM or the TLSTM) while the transport security model (TSM) simply pulls this information out of the shared cache and puts the data into the corresponding ASIs. As a consequence of this approach, a single transport security model can be used with a variety of secure transport models.

5.2 Expected Impact

The software is meant to be a reference implementation for the specifications produced by the ISMS working group of the IETF. The implementation experience has already helped to gain some implementation feedback on different design choices. In addition, the imple-

mentation aims to serve as a research platform for prototyping and evaluating alternate secure transports.

In the longer run, we aim at the integration into the `Net-SNMP` package. However, this turns out to be not realistic before the ISMS working group of the IETF has finished the specifications as the source code changes are relatively large and it would not help ISMS if different versions of implementations of working group drafts are spread around. This is an important consideration since several vendors (e.g., Apple) rely on `Net-SNMP` code for the SNMP agent shipped with their operating system.

5.3 Progress Report

A prototype implementation of SNMP over SSH / TLS / DTLS has been developed at Jacobs University as an extension of the widely used open source `Net-SNMP`¹ SNMP implementation (version 5.3.0.1). For the SSH protocol, the open source `libssh`² C library was used. It contains all functions required for manipulating a client-side SSH connection and an experimental set of functions for manipulating a server-side SSH connection. For the TLS and DTLS protocol, the open source `openssl`³ library was used.

The implementation itself consists of a new C module implementing the TSM (about 300 lines of code) and three additional C modules implementing the transport models SSHTM, TLSTM, and DTLSTM (each one roughly 900 lines of code). The `Net-SNMP` internal API for adding transports worked reasonably well and did not require any changes. The fact that `Net-SNMP` already supports stream transports was convenient. For password authentication, the SSHTM calls the Linux Pluggable Authentication Modules (PAM) [19] library to make it runtime configurable how passwords are checked.

Most of the development time was spend on optimizing the performance of the implementation since the overall latency initially was surprisingly high. In order to optimize the performance of the SSH transport domain, we investigated the influence of TCP's Nagle algorithm as well as the windowing mechanism of the SSH protocol. For the DTLS implementation, we had to investigate how to set the DTLS timers and patch a long known bug in `openssl`. Furthermore, we did run into some limitations of the `Net-SNMP` package we used.

5.3.1 TCP Nagle Interactions

During our initial measurements, we observed that the execution of a `Get` operation over the SSH transport domain required approximately *800ms*. This surprisingly large delay was introduced by TCP's Nagle algorithm, which essentially delays the sending of a TCP segment until either a segment has been filled or the previous segment has been acknowledged. We therefore disabled the Nagle algorithm by setting the `TCP_NODELAY` flag on the agent and on the manager side of the connection. This lead to a significant improvement in the performance of the SSH transport domain as the time required for the execution of a

¹<http://net-snmp.sourceforge.net/>

²<http://0xbadc0de.be/wiki/libssh:libssh>

³<http://www.openssl.org/>

`snmpget` operation went down to *56.5ms*. We further modified the `libssh` library to disable the Nagle algorithm immediately after establishing the TCP connection between the agent and the manager and before any SSH exchanges take place. This further reduced the time required for a `Get` operation to *16.17ms*.

5.3.2 SSH Window Adjustments

The SSH windowing mechanism is used to specify how much data the remote party can send before it must wait for the window to be adjusted. In the OpenSSH implementation, window adjustment messages are only exchanged periodically. During our initial testing of our SSHTM implementatin, we noticed that each message exchanged between the agent and the manager was followed by a window adjustment message. These additional messages introduced significant bandwidth overhead as well as latency overhead for long sessions. As a result the TSM/SSH transport performed worse than the USM transports with respect to latency and bandwidth. In order to optimize the performance, we modified the `libssh` library to send window adjustment messages only when necessary. This improvement lead to better bandwidth and latency performance of the TSM/SSH transport compared to the USM transport.

5.3.3 DTLS Retransmission Timers

DTLS uses a retransmission timer to handle packet loss. The timers are managed internally by the `openssl` implementation. The API simply provides a function to set the timeout. While the API in general is easy to use, the question remains how this timer should be set. Note that SNMP engines usually have their own retransmission logic and hence it is crucial that these retransmission timers play well together. In the TCP based transports, this is less of an issue since TCP adapts the retransmission timers automatically and it is thus safe to turn the SNMP retransmission logic off.

The DTLS and SNMP datagram transports provides more flexibility concerning timers but with flexibility comes responsibility. With the current API, it is non-trivial to implement adaptive timers and hence implementations frequently fall back to fixed interval timers. This is true for both the `Net-SNMP` and the `openssl` implementation we used. In order to let DTLS take care of retransmissions, we have set the DTLS retransmission timer to a fraction of the SNMP retransmission timer but we have kept the SNMP retransmission timer since in principle just the DTLS datagrams carrying SNMP messages could have been dropped and DTLS does not provide a reliable data stream like TCP.

5.3.4 Net-SNMP Limitations

During the implementation, we noticed a number of limitations of the `Net-SNMP` package that are important also for the performance evaluation. The first limitation concerns the `Net-SNMP` retransmission timers, which by default can only be set in the resolution of seconds (with a default of one second). On many Internet links, waiting a second to start a

retransmission is rather ineffective. We changed the source code so that we can specify retransmission timers in milliseconds.

A second and more significant limitation concerns the handling of the `msgMaxSize` SNMPv3 message header field. This header field is of particular importance for the processing of `GetBulk` requests since it provides a hint for the maximum response message size that can be returned to the manager. Unfortunately, the `Net-SNMP` agent code does ignore this hint and thus it is difficult to control the response message sizes generated by an agent. It is even possible that the agent generates responses that do not fit into the buffers provided by the manager and are thus dropped.

The third issue is related to the handling of `contextEngineIDs`. According to the SNMP specifications, an agent may provide access to multiple contexts and the `contextEngineID` and the `contextName` is used to select the appropriate context. A special context engine discovery procedure has been defined for use with the TSM [20]. During implementation, it turned out that the used `Net-SNMP` version actually treats any `contextEngineID` as referring to the local default context.

5.4 Conclusions

The ISMS reference implementation development funded by the EMANICS project has been important to ensure that the specifications developed in the ISMS working group of the IETF are implementable. The work also led to additional specifications, such as the SNMP EngineID discovery document [20], that are necessary to make the technology complete and implementations work. In addition, the implementation allows us to experiment with other secure transport models such as TLS and DTLS.

While the implementation is functional, there is additional work to be done as the working group proceeds. Since the `Net-SNMP` version we started from has been superseded by another major release, additional work is necessary to port our ISMS implementation to the current `Net-SNMP` code base. Since the ISMS implementation creates new dependencies on external libraries, work is needed to create proper cross platform configuration scripts. Finally, there is some work to be done on the integration with `Net-SNMP` command line and configuration file processing.

6 P2PFastSS

6.1 Introduction

Peer-to-peer (P2P) systems have evolved from unstructured systems like Gnutella to structured systems like Chord [?], CAN [?], Pastry [?], or Kademlia [?], which implement efficient distributed hash tables (DHTs). P2P systems can be used to overcome limitations present in centralized systems. Such limitations include low scalability, weak fault tolerance, and poor load balancing.

Basic operations in a DHT are `get(key)` and `put(key, value)`. Data is stored in a DHT by applying a hash function to the key and calling the `put` method. A search is performed using the `get` method. The same hash function is applied to the query. The benefit of using a structured P2P system is that a key lookup typically requires $O(\log n)$ routing steps, where n is the number of peers in the system. However, DHTs have limited support for similarity search, because calculating a hash function on similar keys results in different hash values. DHT-based systems lack support for similarity search using the edit distance metric. Existing solutions are limited to n -gram based substring searches [?] and IR-type ranking based on the Jaccard coefficient [?].

P2PFastSS (fastss.csg.uzh.ch) is a content-based full-text fast similarity search algorithm that uses the edit distance metric and is applicable to the service discovery domain. The edit distance [?] is the minimum number of operations required to transform one string into another, using the operations: deletion, insertion, and replacement. The proposed algorithm can be used on top of any DHT, since it uses only the basic operations `get` and `put`.

P2PFastSS can work with any type of textual information. To test its functionality, abstracts of Wikipedia articles are used as documents. Both the title and the content of these articles did provide a base for similarity search. A number of articles were downloaded from Wikipedia in a structured XML file format and it was then provided to one of nodes in the P2PFastSS network.

6.2 P2PFastSS Architecture

A naive approach to similarity searching is to use broadcasting or flooding based P2P networks. Each peer in the network will receive the query and can locally execute a pattern matching algorithm. However, broadcasting or flooding typically do not scale well. This section describes the algorithms related to similarity search and related work that addresses scalability issues. Similarity search can be used with the edit distance, which is the minimum number of operations required to transform one string into another. An operation is either an insertion, a deletion, or a replacement. The edit distance is calculated using dynamic programming (DP) in $O(pq)$ time where p and q are the lengths of strings being compared. For example, the edit distance between `house` and `mouse` is 1 because the minimum operation to transform `house` into `mouse` is to replace `h` with `m`.

P2PFastSS finds text documents containing keywords similar to the ones included in the search query. Similarity in P2PFastSS is defined as the edit distance k between keywords.

P2PFastSS is built on top of a structured P2P network in order to achieve a lookup time logarithmic in the number of nodes. The algorithm is split into two phases. In the first phase the document is stored and indexed, while in the second phase the similarity search is performed.

6.2.1 Indexing and Storing (First Phase)

In the first phase documents in P2PFastSS are stored using the DHT operation `put(key, value)`. The key of the document is the hash of the document title. The value of such an entry is the document itself (see Table 1). All words included in the document need to be indexed before a similarity search can be performed. Indexing is performed using the following steps.

Table 1: Document index excerpt of node 0x1235

Title (Key)	Document (Text)
Albedo (Doc ID: 0x123)	The albedo of an object ...
Achilles (Doc ID: 0x132)	In Greek mythology, Achilles ...
Paper (Doc ID: 0x238)	Paper is a commodity of thin ...
Testing (Doc ID: 0x321)	This test aims to ...

1. All words in the document are identified.
2. A deletion neighborhood is generated for all words in the document.
3. All neighbors are stored with a reference to the document IDs in which the word appears, using the operation `put(key, value)`. As words may appear in many documents, a key can hold multiple values (a list of references).

In the following example, keys and values of index entries are shown. Table 1 shows the keys and values of a nodes document table. Table 2 shows keys and values of a nodes keyword index, and Table 3 neighborhood entries for test with $k=1$. The key is determined by the hash of the document title (Table 1) or keyword (Table 2). The value contains the text (Table 1), or the keyword and a reference that points to the document (Table 2). If a word appears in several documents, it is mapped to a list of locations.

Table 2: Keyword index excerpt of node 0x1235

Keyword (Key)	Resource and Location
object (hash 0x424)	object, Doc ID: 0x123...
pper (hash 0x927)	paper, Doc ID: 0x238...
wter (hash 0x149)	piper, Doc ID: 0x641...
tes (hash 0x123)	water, Doc ID: 0x583...
	test, Doc ID: 0x321...

During the indexing phase P2PFastSS generates a deletion neighborhood by using the method precalculate and will index all neighbors of word test (Table 2). The messages sent by the searching node in the process of indexing and storing a neighbor in a distributed environment are shown in Figure 17. Node 0x1 wants to index the deletion neighborhood of the word test (hash 0x563), which points to the document with ID 0x321. In messages 1 and 2, it finds the locations appropriate for placing the index entries for the first neighbor tes, which involves two routing queries. Then tes is stored redundantly in message 3. Messages 1 and 2 are sent sequentially, while messages 3 can be sent in parallel. This process continues until all neighbors and the keyword are stored.

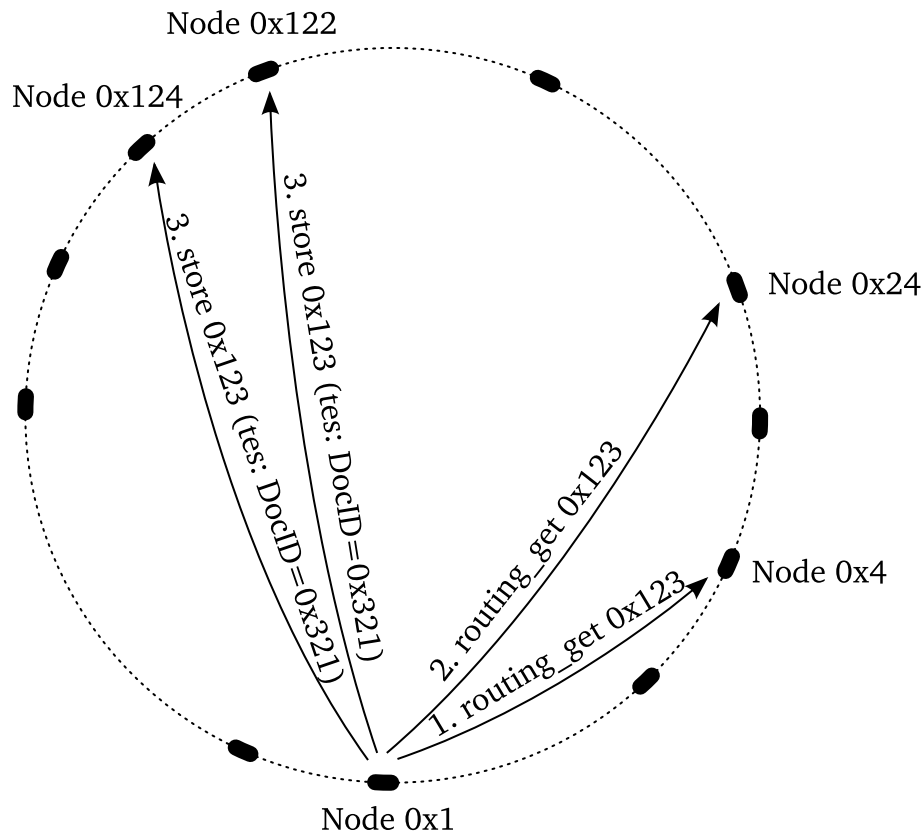


Figure 17: Message diagram of the indexing phase of tes (hash:0x123)

Table 3: Keyword index excerpt of node 0x1235

Keyword (Key)	Resource and Location
object (hash 0x424)	object, Doc ID: 0x123...
pper (hash 0x927)	paper, Doc ID: 0x238...
wter (hash 0x149)	piper, Doc ID: 0x641...
tes (hash 0x123)	water, Doc ID: 0x583...
	test, Doc ID: 0x321...

6.2.2 Searching (Second Phase)

The similarity search is performed in the second phase using the following steps.

1. A node generates a deletion neighborhood from the search keyword.
2. Every neighbor is searched for using `get(key)`. The result contains a document ID and the keyword. Before the document is retrieved using `get(key)`, potential matches are confirmed using DP (see Section II) to check if they do not exceed the given edit distance.
3. The document is retrieved with `get(key)` using the matching document IDs found in step 2.

The message diagram for the search in a distributed environment is shown in Figure 18. Node 0x1 searches for the word `tesa`. Before searching, P2PFastSS generates a deletion neighborhood from `tesa`: (`tes,tea,tsa,esa`) and starts by searching for the first neighbor `tes`. The key for `tes` is `0x123`. First, the node address is found via messages 1 and 2. In message 3, the node containing `tes` is looked up. Messages 1, 2, and 3 are sent sequentially. Finally, node 0x122 returns the Doc ID, which can then be used to retrieve the document containing the term `tes` which is similar to `tesa` (edit distance 1)

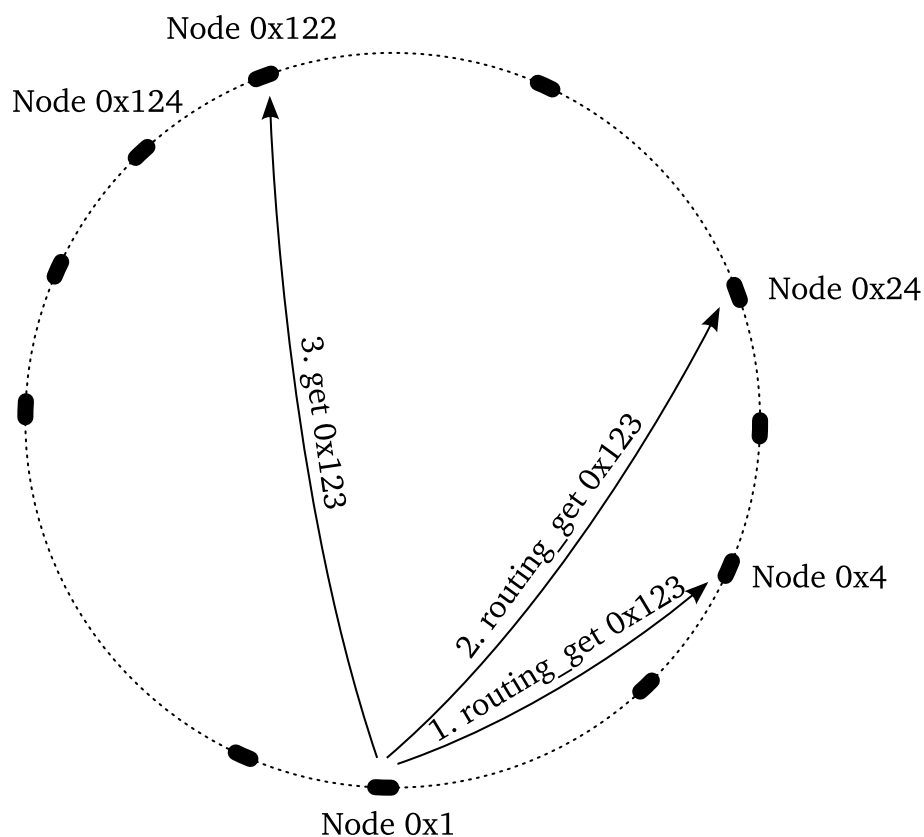


Figure 18: Message diagram of the searching phase of `tes` (hash:0x123)

6.3 P2PFastSS Changes and Extensions

P2PFastSS does not use the data itself to perform similarity search — it needs to build an index of the data before it is able to search. Besides searching, any node has storing and indexing capability. However, creation of the index was not distributed; only one node was responsible for it, while others were idle. Therefore, it could take a long time to index large datasets. This behaviour has been changed, allowing any node to distribute indexing effort randomly to other nodes. With distributed indexing, an initiating node is still needed, since a specific node must coordinate and distribute articles for other peers to index.

Distributed index works as follows: 1. an initiating node extracts articles from the XML file and uses the DHT put operation to write them into the DHT, effectively sending it to a random peer in the network; 2. any nodes that receive a full article must create an index with all words in the article, plus their deletion neighborhood for the constant edit distance; 3. those peers, after building the index, use the put operation to store every generated deletion neighborhood into the DHT.

Once the deletion neighborhood was computed, a key containing the result of a hash function applied on the articles title was computed and the article was stored into the DHT by using `put(key, article)`. A normal DHT operation, `put` routes and stores the article in a node with node ID closest to the key. A slightly different procedure was used to store the indexed data. For each deletion neighbor of a given word, a special object, called `KeyAndKeyword`, was created. This object contained the original word, from which the deletion neighbor was generated, and the title of the article from which the word belongs. The reason behind this approach is that different words can generate equal deletion neighbors; hence, a way to connect the index data with its original target data is needed. Once the `KeyAndKeyword` object was created and initialized, it was put into the DHT. There was no logical separation in code for article storing in DHT and their indexing: it was all done by a single method `storeArticles()`. The path to the XML file could only be passed as argument before running the program: once the XML file was processed and the articles indexed, no other articles could be added while the program was running. The software was extended with a web interface which allowed selecting of the XML file to process, manual insertion of articles into the system, submitting of search queries and displaying the search results.

Each node had separated data structures for managing the storing procedure of articles and of the index data. Both structures used implemented the `Map` interface. For the articles, two `ConcurrentHashMaps` were used, together with following Genenrics as (key, value) pair: (`BigInteger`, `Map<String, Object>`). The reason behind this “map-in-map” approach was to allow the user to specify an additional domain as a `String`, allowing storing more objects for a single key.

A first action was to logically separate the various tasks, by using different Java methods for each of them. So the `storeArticles()` method was replaced by removing the code that was not involved with article storing. A new method for article indexing purpose, `indexArticle()`, was created. The method parameters are the article to index, and an integer to indicate the edit distance desired for the indexing. This method creates `KeyAndKeyword` objects with generated deletion neighbors and passes them to the DHT for storing. To avoid blocking a node while it is indexing articles, a dedicated thread, called `IndexRunner`, is added to the P2PFastSS node. It runs in parallel with the node and takes care of indexing. When there are no articles to index it can be put in sleep mode. Different solutions

were considered regarding how could a node could differentiate what type of objects it stores in his map. Initially, message payload analysis and reengineering of the class managing the article's map (StoreMultiMapMemory), to allow callbacks on the nodes main class, was considered, but those solutions would have been complex and inelegant. A much more elegant solution is to slightly modify StoreMultiMapMemory, to allow listeners to be added to it. A last update to this class code was to notify all its listeners whenever an article was going to be stored on the map. Finally, node's constructor was modified, so that an anonymous listener is created and added to the node's StoreMultiMapMemory. This way, whenever an article is stored, eventual listeners are notified. They then send the article to the IndexRunner for indexing and putting the index data in the DHT.

6.3.1 Disk-based Storage

The former implementation of P2PFastSS used peer's volatile memory to store the data. While this allowed fast access to the stored information, it had two main drawbacks:

1. Given the nature of RAM, once the P2PFastSS application running on a node was closed (intentionally or not), all article and index data stored in that node were lost. If the whole P2PFastSS network was shut down, all articles would have to be processed again to return to the previous state.
2. Index data size tends to be multiple times the size of the text. In [?] it is stated that a 388 KB dictionary results in an approximate 100 MB index for an edit distance of 2, which is more than 250 times larger. This situation causes the P2PFastSS application to run out of system memory.

By implementing a disk-based storage, these drawbacks could be countered: nodes could rejoin the P2P system after failure, and the system could resume after a shutdown, without need to re-index all the data.

Before implementing the disk-based storage for P2PFastSS, an important decision had to be made. It concerned the choice of storage system to be used: the file system (FS) or a database management system (DBMS). Table 4 shows some key benefits and drawbacks considered for both systems.

Benefits	Drawbacks
<p>FS Simplest approach: Java Serialization could be used. Special file format could be used. Can be compressed. (DB also) Java Serialization is slow and sensitive to versioning; size of serialized data usually larger then objects size in memory. If special format used: parsing of file is necessary. Read / write access must be managed and synchronized, as both can happen concurrently</p>	<p>DBMS Standardized Language (SQL). Open source DBMS software available. Complex queries possible DBMS responsible for concurrent read/write access Flexibility in accessing database table fields. DBMS responsible for data integrity CPU overhead from DBMS threads Memory overhead from DBMS threads</p>

It was decided to use a relational database manager. HSQLDB was used for the following reasons: It is written in Java and offers JDBC APIs, it is available under BSD license, library size is small (around 600 KB for the standard edition), no need to start the server in separate JVM (can run in In-Process mode).

Table 4: Benefits and drawbacks of File Systems and Database Management Systems

	Benefits	Drawbacks
FS	<ul style="list-style-type: none"> - Simplest approach: Java Serialization could be used. - Special file format could be used. 	<ul style="list-style-type: none"> - Java Serialization is slow and sensitive to versioning; size of serialized data usually larger than objects size in memory. - If special format used: parsing of file is necessary. - Read / write access must be managed and synchronized, as both can happen concurrently
DBMS	<ul style="list-style-type: none"> - DBMS responsible for data integrity. - Open source DBMS software available - Complex queries possible - DBMS responsible for concurrent read/write access. - Flexibility in accessing database table fields. - Standardized Language (SQL) 	<ul style="list-style-type: none"> - CPU overhead from DBMS threads - Memory overhead from DBMS threads

6.3.2 Keyword Relevance

To test the correct running of the P2PFastSS system and to allow a simple view of the search results, a simple web interface is used; it allows the user to select XML files for indexing, to perform similarity search and to display the results. However, the results weren't sorted: they were displayed in the order the various nodes answered to the search queries. This caused situations in which results with an exact match (with edit distance 0) were displayed after some results with larger edit distance. In order to sort results by relevance, a numerical value representing the relevance of a keyword in an article has to be calculated.

A "relevance function" was added, which assigns, at indexing time, a numerical value to each deletion neighbor calculated. The factors that influence the relevance score are edit distance, whether the keyword appeared in the article title. The relevance score is saved with each generated deletion neighbor and is used as the base for sorting the search results before showing them on the web interface. The list below enumerates some important properties that have impact on a keyword's relevance: Edit distance (the lower, the greater relevance), number of occurrences of the keyword in the article, length of the keyword (in characters), total words contained in an article, occurrence in article title (if the keyword appears in the title the score should be increased).

Since the application must store, with each deletion neighbor of an indexed word, its relevance score in an article, the KeyAndKeyword class is modified to carry the additional relevance variable.

6.3.3 Paging of Search Results

In the previous version, all the results were displayed in a single page. Whenever a search query produced many search results, this page would result higher than the user's screen, so he or she would have to scroll the page to check all the results. If the results had to be verified one at a time, then the user had to visually keep track of the last verified result, which could be tedious and frustrating for the user. Another issue was the results page loading time, where a page displaying many results would take more time to be visualized. For this reason a paging feature was implemented, so that the search results would have been separated into multiple sets.

Only one set of results would be displayed on the results page at a given time, and the user can decide to navigate back and forth through the sets. The sets would have to be sorted by relevance so that those displayed first would contain the results with higher relevance score than those displayed later. The user browses the search results by selecting the previous or next link. Navigation through the results should not imply new search queries. The first page of search results does not show the previous link, while the last page does not show the next link. A default number of results per page is predefined and if the result set is smaller or equal to it, both links are not displayed.

To obtain the sorting of search results, they need to have an ordering relation. The most natural ordering relation is their keyword relevance score. By the time this particular implementation has begun, the keyword relevance function was already implemented and available. But the `KeyAndKeyWord`-objects still could not be compared between each other. To make them comparable, it was decided to modify `KeyAndKeyWord` class so that it would implement the `Comparable` interface. Consequently, the method `compareTo()` is implemented, making `KeyAndKeyWord`-objects comparable by relevance. A new collection type is used for containing the results of search queries to avoid manual sorting. The Java class `TreeSet` is chosen for this task because it automatically sorts the elements it contains by using their `compareTo()` method, and updates the order if needed when elements are added or removed. Finally, the code for submitting queries was logically separated from the code responsible for displaying the results. The results of a search query are temporarily saved in a `TreeSet` object and this object is then iterated to obtain the results for the displayed page. Iteration of the `TreeSet` does not produce new search queries or additional network traffic.

6.3.4 Preview of Search Results

The visual output for the search results displayed in the web interface consisted only of the articles title and a numerical hash value of it. If the title was not informative enough for the user, he had to open several articles, until the searched article was found. If the articles were not on the node that user was using, then the node had to fetch many of them from the DHT, thus generating network traffic. It is assumed that, if the user was given more information about the search results, he or she would identify the searched article (or other type of resource) faster and it would generate less network traffic.

To present better contextual information to the users, each result must include a segment containing the keyword in the article. It was decided to extend the software so that when

a keyword is indexed, its position in the article is also identified and a short segment of text containing the keyword, defined as the preview string, is saved together with the index results.

The KeyAndKeyWord class is furthermore extended to allow saving of the preview string inside of it. At index time, for each generated deletion neighbor, a portion of the articles text, containing the original keyword and a given number of surrounding words from the left and from the right side, is saved as the preview text. To allow the preview string to be saved on disk, the database table for index data is extended.

The web interface is modified to show the preview text of the article, in addition to the articles title. Before actual displaying, the search results are processed, the keyword is identified in the preview string and HTML bold tags are applied to it, as Figure 19 shows, to make the keyword even more visible to the user.

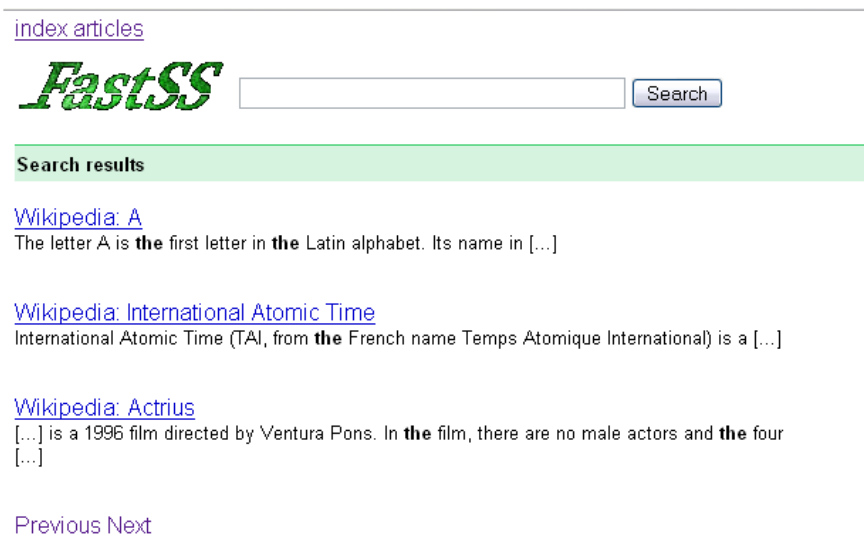


Figure 19: P2PFastSS screenshot

6.4 Impact Evaluation

P2PFastSS and TomP2P are in an early stage and the first version has been released to the project site (<http://tomp2p.csg.uzh.ch/>). The project page is expected to be a source of knowledge for similarity searches in P2P networks. In future, mailing lists and forum will be added for public discussions about the development of similarity searches in P2P networks. Current developers of P2PFastSS are: Thomas Bocek (UZH, lead), Fabio Hecht (UZH), and Dalibor Peric (UZH).

7 NDPMon packaging and distribution

7.1 Presentation

Please refer to section 4.1 for a presentation of the NDPMon framework.

7.2 Expected Impact

Ensuring standard organization and availability for most common operating systems will increase the number of potential users and extend the community around the tool, bringing more feedback and visibility. A binary distribution, especially if the package is integrated in the official repositories, will also increase these two points. Finally, we expect to ensure the durability of the tool thanks to these operations, by developing the developers community, which will emerge from these tasks.

7.3 Progress Report

In this section, we will present the progresses and new features brought to NDPMon since the beginning of the EMANICS support campaign regarding organization and distributions.

File System Hierarchy Standards

The first step towards packaging and distribution is to meet the standards especially the File System Hierarchy Standards (FHS - <http://www.pathname.com/fhs/>), in order to place all the files in their right place, and ensure the compatibility. The default paths used by NDPMon are compliant with the FHS since version 1.3 from the 10 September 2007.

By default, the files are installed in:

- /usr/local/ndpmon: sources, plugins and scripts
- /usr/local/etc/ndpmon: configuration file
- /usr/local/share/man/man8: manpage
- /usr/local/sbin: executable
- /var/local/ndpmon: variable data (discovery history, neighbor cache, alerts)
- /etc/init.d: startup script

All these paths can be changed via the configure script before building the tool.

Binary Packages

In order to ease the distribution on the tool, we enabled binary distribution for Debian and FreeBSD.

NDPMon is available as a FreeBSD port (<http://portsmon.freebsd.org/portoverview.py?category=net-mgmt&portname=ndpmon> and <http://www.freebsd.org/cgi/cvsweb.cgi/ports/net-mgmt/ndpmon/>). Janos Mohacsi from HUNGANET currently maintains this port.

Debian packages are also available for AMD64 and i386 architectures via the tools main download site (https://sourceforge.net/project/showfiles.php?group_id=178666&package_id=247867). These packages are maintained within the team.

SourceForge Project

NDPMon is distributed under the LGPL licence as a SourceForge project. The project itself is available at the URL <https://sourceforge.net/projects/ndpmon/>. NDPMon also has an official WEB site at the URL <http://ndpmon.sf.net/>. Having a SourceForge project ensures the tools visibility, and eases the creation and management of the developers community. At the moment, the NDPMon community is composed of 4 members:

- Frederic Beck (INRIA/LORIA, France) - Maintainer
- Thibault Cholez (INRIA/LORIA, France)
- Pierre Chifflier (INL, France)
- Janos Mohacsi (NIIF/HUNGARNET, HUNGARY) - FreeBSD Port Maintainer

The community also welcomed the contributions of:

- Steffen Strueber (TU Clausthal, Germany): new alerts systems with pipes, corrected warnings
- Maik Frchtenicht (TU Clausthal, Germany): ndpmon2html

In the SourceForge project page, all the information about the tool is available, such as statistics on the projects activity, project details, screenshots SourceForge is also hosting the sources repository and code version via a Subversion repository, which can be browsed via HTTP. Besides this source access, the tools releases are available for downloading on both the WEB site and SourceForge via several packages:

- ndpmon: the sources of the tool
- ndpmon-extras: ndpmon2html and statistics
- ndpmon-web-interface: the WEB interface

- ndpmon-packages: binary distribution

For support, two mailing lists are available:

- ndpmon-users: general discussions about the tool and help to the users
- ndpmon-devel: discussions between developers and Subversion reports

NDPMon also has 2 forums and a bug tracker. All documentation and how-to are available on the official WEB site.

8 OpenDiameter Packaging and Distribution

University of Zurich (UniZH) is actively involved in providing OpenDiameter binary packages for Linux distributions Ubuntu [?] and Debian [?].

8.1 OpenDiameter

OpenDiameter [?] is an open-source implementation of the Diameter protocol and several applications based on top of it. The implementation of the protocol is based on RFC3588 [?] designed by the IETF's AAA Working Group. The source code of the OpenDiameter software is available under the combination of Lesser GNU Public License [?] and GNU Public License [?]. The base protocol implementation is available as a C++ library and has currently support for Linux, BSD and Windows systems. It relies on ACE [?] to provide system level abstraction for all supported systems.

The libraries that OpenDiameter provides are:

- *libdiamparser*: Diameter message parser library (with XML dictionary support), with capability of user-defined AVP type parsers. As all configuration options are kept in XML files this library offers support for reading those files.
- *libdiameter*: Diameter core engine library with base accounting support. Based on this library further Diameter applications can be built. The library includes a light authentication and authorization application as well as an accounting application.
- *libdiameter-eap*: The Diameter EAP (Extensible Authentication Protocol) Application library may be used to build authentication applications based on EAP.
- *libeap*: The EAP library implements different EAP payload types and their handling.
- *libdiameter-nasreq*: The Diameter NASREQ Application library implements the NASREQ Application. It provides AAA services for dial-in PPP users and is the next generation replacement for the RADIUS protocol.
- *libdiameter-mip4*: The libdiameter-mip4 library provides a C++ API to Diameter MIP v4 (Mobile IP v4 Protocol) Application. The library implements the specification defined in draft-ietf-aaa-diameter-mobileip-20.txt.
- *libpana*: It implements the Protocol for carrying Authentication for Network Access (PANA). It is offered in IPv4 and IPv6 support for Linux, FreeBSD and Windows.

8.2 Expected Impact

Installation of OpenDiameter libraries from source may be a lengthy process due to its dependencies on different libraries. Having the OpenDiameter libraries as binary packages helps developers by offering an easy and fast way of installation. The interest for the OpenDiameter binary packages was observed on the CSG@IFI webpage hosting those

packages. The number of page hits is constantly increasing, currently having about 200 monthly hits. A set of EU-funded projects (such as Daidalos [?], EC-GIN [?], SmoothIT [?]) are using OpenDiameter for their research and are expected to make use of the OpenDiameter packages. Other EU-funded projects already used these packages (such as Akogrimo).

8.3 Progress Report

The OpenDiameter binary packages are already available on the CSG@IFI webpage [?]. The packages are based on the source code available from the OpenDiameter developers. Whenever a new version of OpenDiameter is released it is packaged for the most recent Debian and Ubuntu distributions.

OpenDiameter libraries are organized in three packages:

- *libdiameter1.0.7*: Includes all the OpenDiameter shared libraries. Those shall install under */usr/lib*.
- *libdiameter-dev*: Includes OpenDiameter header files used for development of applications on top of OpenDiameter. The files shall install under */usr/include/opendiameter*.
- *opendiameter*: Includes the *aaa/nas/pac* example binaries and respective configuration files. These examples are the one that OpenDiameter developers included in the source package for testing OpenDiameter.

Currently UniZH offers OpenDiameter packages for several Linux distributions and several OpenDiameter versions. The available packages by February 20th, 2008 are:

- Ubuntu Gutsy - OpenDiameter 1.0.7-h and 1.0.7-i
- Debian Etch - OpenDiameter 1.0.7-h and 1.0.7-i
- Ubuntu Dapper - OpenDiameter 1.0.7-h
- Ubuntu Breezy - OpenDiameter 1.0.7-h
- Ubuntu Hoary - OpenDiameter 1.0.7-h
- Debian Sarge - OpenDiameter 1.0.7-h

8.4 Debianizing Open Diameter

In order to "debianize" the OpenDiameter software, the following steps had to be undertaken:

- First, the required Debian tools had to be installed for creating the packages, using the following command: `apt-get install dpkg-dev dh-make fakeroot`.

```
Source: opendiameter
Priority: optional
Maintainer: David Hausheer <hausheer@ifi.unizh.ch>
Build-Depends: debhelper (>= 4.0.0), autotools-dev, libssl-dev,
  libxerces26-dev, libace-dev, libboost-dev
Standards-Version: 3.6.1
Section: libs

Package: opendiameter
Section: net
Architecture: any
Depends: ${shlibs:Depends}
Description: Diameter binaries and documentation

Package: libdiameter-dev
...

Package: libdiameter1.0.7
...
```

Figure 20: control file

- Afterwards, a Debian package template was generated using the `dh_make` command, following the instructions given in the Debian Package Maintainers' Guide [?]. This resulted in a folder called `debian` within the `OpenDiameter` folder.
- As a next step, the corresponding configuration files within the `debian` folder had to be edited, namely the `control` file and the `rules` file. The `control` file is depicted in Figure 20, while Figure 21 shows the commands to configure the package in the `rules` file.
- Now the package installation files had to be created for the 3 different packages. Those are depicted in Figure 22.
- The `OpenDiameter` copyright notice was included in the `copyright` file and a `changelog` file was created, which has to be updated for every new version. A `changelog` entry example is shown in Figure 23.
- Finally, the package was compiled using the command `dpkg-buildpackage -rfakeroot`. Any rebuild can be done with the command `fakeroot debian/rules binary`.

8.5 Debian Repository for Open Diameter Packages

In order to create a Debian repository to download the Open Diameter packages, the following steps were necessary, as described in [?]:


```
config.status: configure
dh_testdir
CFLAGS="$(CFLAGS)" \
XERCESROOT=/usr/include/xercesc \
ACE_ROOT=/usr/include/ace \
BOOST_ROOT=/usr/include/boost \
./configure \
--host=$(DEB_HOST_GNU_TYPE) \
--build=$(DEB_BUILD_GNU_TYPE) \
--prefix=/usr \
--enable-shared \
--enable-static
```

Figure 21: Commands to configure the package in the rules file

```
libdiameter-dev.dirs:
usr/lib
usr/include
usr/share/doc/libdiameter-dev

libdiameter-dev.install:
usr/include
usr/lib/lib*.a
usr/lib/lib*.so
usr/lib/*.la

libdiameter1.0.7.dirs:
usr/lib
usr/share/doc/libdiameter1.0.7

libdiameter1.0.7.install:
usr/lib/lib*.so.*

opendiameter.dirs:
etc/opendiameter
usr/share/doc/opendiameter

opendiameter.install:
usr/bin
usr/etc/opendiameter    etc/opendiameter
usr/share/opendiameter/doc    usr/share/doc/opendiameter
```

Figure 22: Package installation files

```
opendiameter (1.0.7-f-1) unstable; urgency=low
```

```
* Initial Release.
```

```
-- David Hausheer <hausheer@ifi.unizh.ch> Thu, 22 Feb 2008 13:59:03 +0100
```

Figure 23: Changelog entry example

```
dists
+-etch
  +-main
    |-binary-i386
    +-source
```

Figure 24: Folder tree layout

- A folder tree layout had to be created as shown in Figure 24.
- An archive config file `apt-ftparchive.conf` had to be generated. This is depicted in Figure 25.
- For every distribution a release config file `apt-<distr>-release.conf` was created. This is shown for Debian etch in Figure 26.
- To generate the repository, the following command was executed: `apt-ftparchive generate a`
And for every distribution as shown for etch: `apt-ftparchive -c apt-etch-release.conf
release /home/csg/debian-csg/dists/etch > /home/csg/debian-csg/dists/etch/Release`
- Finally, to add install the OpenDiameter packages, the following line has to be added to `/etc/apt/sources.list`: `deb file:/home/csg/debian-csg etch main`

```
Dir {
  ArchiveDir "/home/csg/debian-csg";
};

BinDirectory "dists/etch/main/binary-i386" {
  Packages "dists/etch/main/binary-i386/Packages";
  Contents "dists/etch/Contents-i386";
  SrcPackages "dists/etch/main/source/Sources";
};

Tree "dists/etch" {
  Sections "main";
  Architectures "i386 source";
};
```

Figure 25: Folder tree layout

```
APT::FTPArchive::Release::Origin "debian-csg";
APT::FTPArchive::Release::Label "debian-csg";
APT::FTPArchive::Release::Suite "etch";
APT::FTPArchive::Release::Codename "etch";
APT::FTPArchive::Release::Architectures "i386 source";
APT::FTPArchive::Release::Components "main";
APT::FTPArchive::Release::Description "Etch debian packages for csg";
```

Figure 26: Release config file apt-etch-release.conf

9 An online catalogue of available software for network management

This chapter describes only the work done after changes made into the Emanics Repository & Inventory application to use together with the Simpleweb.org repository (hosted at UT), what solved the database replication and coherence maintenance problems. All the work done before was described in the D6.1 and D6.2 - Emanics Deliverables.

For the Emanics phase II the following objectives had been pointed out within the software database extension joint UT and PSNC proposal:

- Any needed work related to current maintenance of the repository database and GUI web-application;
- Research and development works extending the existing repository's functionality, divided into the following two tasks:
 - Base extension for annotation support;
 - Classification of existing registered packages.

According to the aforementioned objectives during the Emanics phase II PSNC has performed some necessary maintenance works concerning Emanics repository web-application and database. PSNC made a few corrections in the application, which have been causing small problems with repository visualization in some of the web-browsers using another font set. All these corrections were thoroughly tested by PSNC before putting into practice in the official project web site.

In the last quarter of the year 2007 PSNC has initiated the discussion and cooperation with all project partners to choose the best way for repository extension. During this discussion and opinions exchange we proposed also our own technical and organisational ideas and schemas how to attain the intended objectives. As a result a proposal of about 10 the most important software packages was made which are considered as worth of an additional annotation in the existing repository. PSNC proposed the initial version of network management packages list, which was extended by project partners and finally accepted. At the end of this process PSNC proposed the following list of the best software packages designated for an additional annotation in the repository:

- cacti
- rrdtool
- net-snmp
- ntop
- wireshark
- cfengine

- nfsen
- nagios
- nmap
- flowtools
- fprobe

In addition to the above proposal we also suggested including several tools developed or ported by EMANICS partners especially in the scope of WP6 T6.3 activity (e.g. Ponder2, SCLI). As a result of our proposal the database hosted at UT was properly updated and the new list of the best and commonly used management software packages selected from the Emanics repository and designated for an additional users annotation is now available at the project's official web site: http://emanics.org/component/option,com_dbquery/Itemid,93/

The example view of this web page is presented below on Figure 27.

At the end of year 2007 PSNC designed, developed and tested the first version of automatic software repository classification. This new functionality of the Emanics repository performs automatic software assignment to the standard task's categories. This allows users to look through the repository not only using the search engine but also as an on-line catalogue of software divided into different functionality's categories. PSNC proposed to divide this software packages into the five categories in pursuance of the standard described in the TMN/Network Management Model proposed by the ITU-T and ISO organizations. These standard categories are known as „FCAPS” which stands for Fault, Configuration, Accounting, Performance and Security. The first programming effects of such new functionality were available to local tests at the beginning of year 2008. Then, after these tests and needed corrections PSNC implemented this functionality also in the operational and official Emanics web site in February 2008. It is publicly available at: <http://emanics.org/content/view/122/146/> and the example view of this web page is shown on the Figure 28.

Below, an example of one out of the five categories is also presented to show the potential of the new Emanics repository functionality. The part of “Performance management” group is shown at Figure 29.

PSNC has also updated the “Emanics Inventory” table to reflect properly the latest database changes coherent with our proposal of the best software list, described earlier. The current view of this inventory is shown below on the Figure 30.

All the new and old functionalities of the software repository, introduced already in its operational version, were fully integrated by PSNC into the official EMANICS web site and are available publicly at the “Software” position of the main menu or via the direct link: http://emanics.org/component/option,com_dbquery/Itemid,93/

The screenshot shows the EMANICS website in a Mozilla Firefox browser. The page title is "EMANICS - European Network of Excellence for the Management of Internet Technologies and Complex Services". The URL is "http://emanics.org/component/option,com_dbquery/Itemid,93/".

The main content area is titled "Welcome to Emanics Repository of Network Management Software". Below this, it states: "The best software packages listed in the Emanics public repository are selected for you below".

Name	Source	Description	More
Cacti	Ian Berry	RRD Front-end	More
Cfengine	Oslo University College	Unix configuration management and anomaly detection software. Widely used.	More
Flowtools	Mark Fullmer	Software for collecting and processing NetFlow data	More
Fprobe	Fprobe team	NetFlow probe	More
libsmi	TU Braunschweig	A library to access SMI MIB information	More
Nagios	OpenSource	Host and service monitor.	More
net-snmp	Univ of California, Davis	Various tools relating to the Simple Network Management Protocol	More
NfSen	NfSen team	Graphical front end for nfdump	More
nmap	Nmap team	A utility for network exploration or security auditing	More
ntop	Luca Deri	A tool that shows the network usage, similar to what the popular 'top' Unix command does.	More
RRDtool	Tobias Oetiker	RRDtool as a reimplementation of MRTGs graphing and logging features.	More
scli	IUB	Scli addresses the need for small and efficient command line utilities to monitor and configure systems.	More
Wireshark	Gerald Combs	Wireshark is a network protocol analyzer for Unix.	More

Below the table, there is a search bar labeled "Search public repository".

Hints on functionality access:

Unregistered users have access only to the Search repository functionality.
Registered users have access also to the Add software functionality.
Only registered users with SPECIAL privileges have additional access to the edit/update functionality. The "EDIT" button is accessible after pressing the "More" button for the selected tool.

At the bottom, the copyright notice reads: "© 2005-2008 EMANICS - European Network of Excellence for the Management of Internet Technologies and Complex Services Participants. Design by Andreas Viklund".

Figure 27: The new list of the best management software packages selected from the Emanics repository and designated for an additional annotation.



Figure 28: The main web page of the Emanics Repository Software Classification.

EMANICS - European Network of Excellence for the Management of Internet Technologies and Complex Services - Mozilla Firefox

http://emanics.org/component/option,com_dbquery/Itemid,150/

Information Society Technologies

EMANICS

Welcome!

About

News

Newsletter

Activities

PhD Theses

Documents

Events

Software

Emantics inventory

Search repository

Classification

Fault

Configuration

Accounting

Performance

Security

QoS Management

Call For Papers

Links

Partners

Contact

Site Map

Members Area

Performance management software packages selected from the Emanics repository

<< Start < Prev 1 2 3 Next > End >>

Results 1 - 20 of 44

Display : 20

Name	Source	Description	More
AdventNet ManageEngine OpUtils	AdventNet Inc.	Comprehensive, Web-based and Quick Troubleshooting Tools for IT Administrators	More
AdventNet QEngine	AdventNet Inc.	To test web applications, web services, schedule tests, track issues and features	More
Alchemy Eye	Alchemy Lab	Server (TCP/IP, ICMP, IPX/SPX, Oracle, MS SQL, NT EventLog, SQL query, HTTP URL) monitoring tool	More
Analyse It	Mechanism Software	Network utilization reporting tool	More
APG (Automated Performance Grapher)	Watch4Net	Performance and SLA reporting	More
Argus Monitoring System	argus.tcp4me	System + Network Monitoring	More
Aruba	Valencia Systems	Network performance and service level management solutions.	More
Bandwidth Controller	Closed	Network Traffic Control Software	More
Chariot	NetIQ	Application layer traffic performance measurement.	More
Chroniker SNMPwatch (old: CleverEye)	Nrg Global	SNMPwatch monitors any SNMP enabled device.	More
CNApro	Generix Ltd.	Wire Based Network Management	More
CommandCenter-NOC	Raritan Americas Inc	Performance monitoring, asset management, security monitoring, traffic/bandwidth analysis and reporting for multiple environments in one virtual or hardware appliance.	More
eNMS Suite	Tavve	Web-based, real-time network management software for fault management, performance reporting, and remote network management.	More
ET/BWMGR - Bandwidth Manager	Emerging Technologies	Bandwidth management product for ISPs and corporate networks.	More
Infosim	close source	Service Level Management Software	More
Intelica IP Intelligence	Intelica Networks, Inc.	Monitor, Analyze and Report IP Network	More
IP Traffic Test & Measure	Omnicor, Inc.	Test, Measure, Stress and Configure IP Networks and Devices	More
LoriotPro	LUTEUS	Network and system management and monitoring software	More

http://emanics.org/component/option,com_dbquery/Itemid,150/

News RSS

[RSS 2.0](#)

CFP RSS

[RSS 2.0](#)

Conferences RSS

[RSS 2.0](#)

Podcasts

[PODCAST](#)

Figure 29: An example view of tools classification for the “Performance management” category.

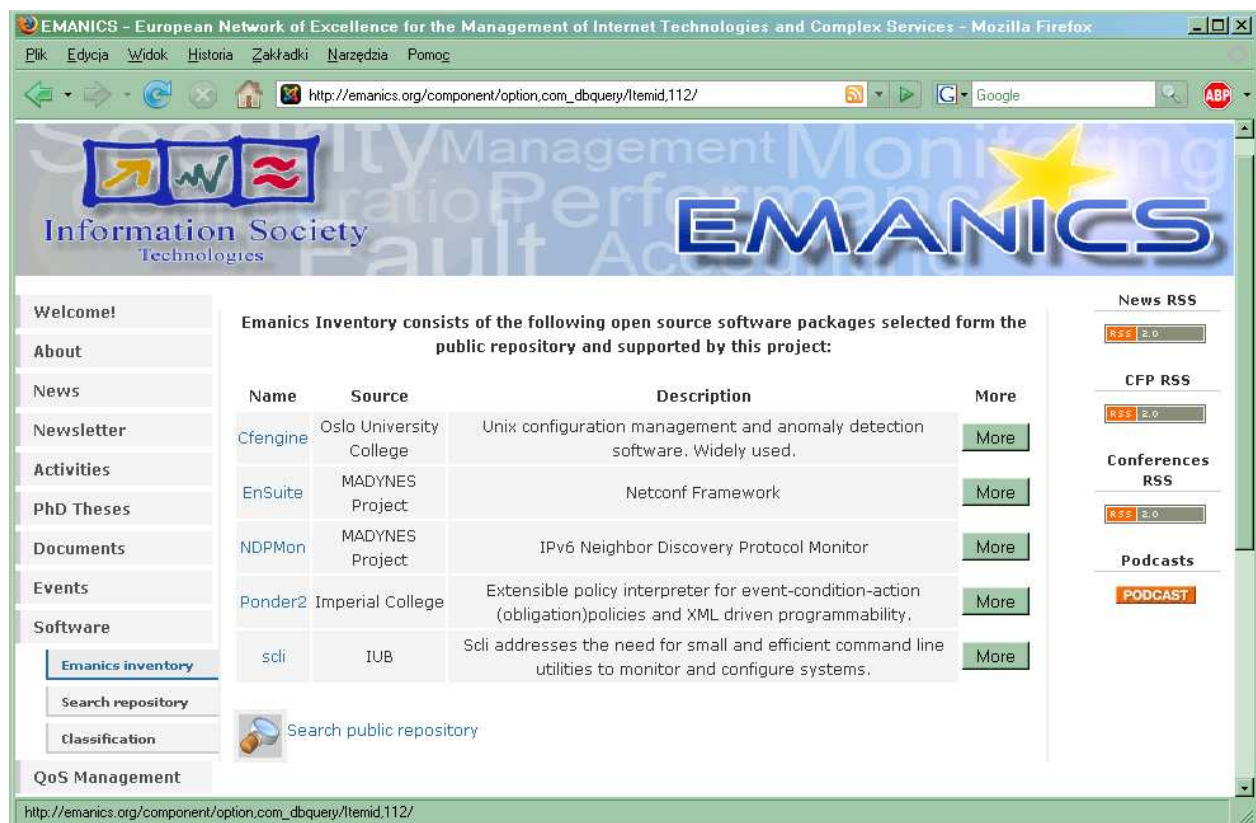
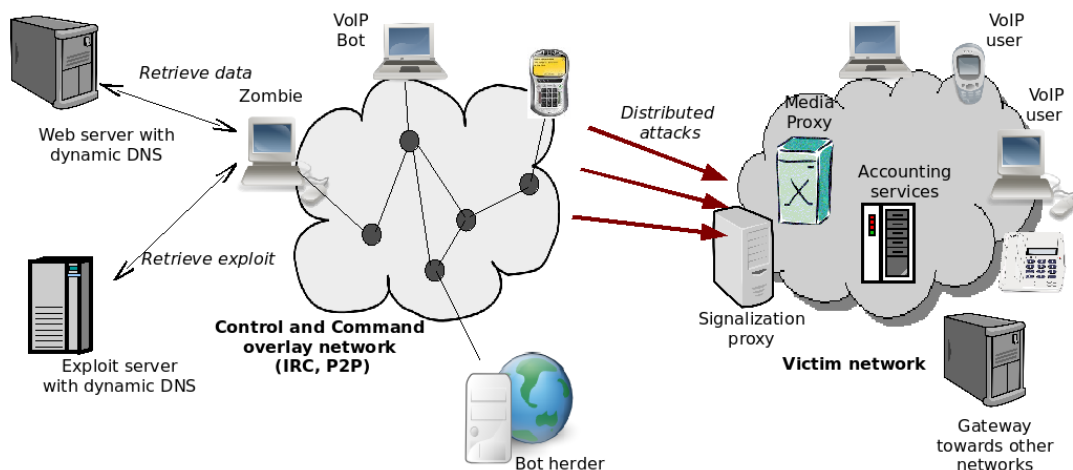


Figure 30: The updated Emanics Inventory web page.

Figure 31: VoIP botnet overlay network



10 VoIPbot

10.1 Description

A bot, short for “robot” describes a program that executes commands when it receives specific input from an external master. A botnet is a large network of devices (usually end-hosts) where bots are installed. These devices called zombies are infected by malware and compromised for the benefit of hackers that can rent them without any knowledge of their real owners. The tenants orientate these bots to send massive SPAM or to launch denial of service attacks. With the growing of the VoIP technology, botnets are expected to extend their activities to attack VoIP services and infrastructures. Values of interest within a VoIP enterprise domain includes signaling and media infrastructure, accounting directories, PBX services (voice mailboxes, gateways) and individual user accounts. Compromising VoIP applications constitutes a bridge to bypass security mechanisms and attack internal networks. Attacks can be transmitted across gateways to integrated networks like mobile and traditional telephony ones. A typical overlay network of a “VoIP botnet” is depicted in Figure 31. We aim, by providing a VoIP bot tool to build a framework of assessment for VoIP domains and VoIP defense solutions. Our bots are not equipped with propagation mechanisms and they don’t represent a real threat value. They are used to emulate good or malicious profiles of VoIP users. In the way we design our bots, we strive to provide a large set of attack scenarios , the most important are described in the following paragraphs:

SPIT

SPIT or SPAM over Internet Telephony refers to unsolicited calls intended for advertising and social engineering. In a SPIT scenario, the attacker asks the bot to deliver an audio record to one or more destination URI. Similarly to e-mails being used in SPAM, URIs can be collected by web crawlers or in result to a VoIP domain enumeration. The bot manufac-

tures an INVITE request carrying an SDP body given arguments like the destination URI, its IP address, its RTP port, the codec to be used and other media attributes. When the call is answered, the bot retrieves the audio record from the location as supplied by the attacker (e.g. from a URL or a local file in the compromised machine) and stream it to the callee.

Flooding

Flooding attacks target the signaling plane elements (e.g. proxy, gateway, etc.) with the objective to take them down or to limit their quality, reliability and availability. Flooding attacks can be categorized regarding their destination and their strategy. Whether the attack is destined to a valid URI in the target domain, a non-existent URI in the target domain, a URI with an invalid domain or IP address, an invalid URI in another domain, or a valid URI in another domain, different damages are produced. The strategy is related to the nature of messages used during the attack: legitimate, malformed (carrying some exploits), invalid (non-compliant to the SIP standard), and spoofed SIP messages (spoofed `From` and `Contact` header). Other attacks are targeted against the authentication process by using messages carrying valid `nonces` and requiring the target to compute digests which overwhelms its CPU capacity [?]. In a flooding scenario, the bot starts a thread which sends continuously request messages (INVITE or REGISTER) given the destination, the duration, the timing and the strategy as supplied by the attacker. Distributed flooding attacks can be easily organized by involving and synchronizing a number of bots.

Enumerating

Enumerating is the process of discovering valid SIP extensions (or URIs) in a SIP domain. Enumerating is usually preceded by a port scan to identify existent SIP proxies and user agents. The standard port used by SIP is 5060. The first step of enumerating is to identify if the SIP service is really running on that port or not and what type and version of server is there. This is done by sending a simple OPTIONS message and interpreting its response. The searched information is usually found in the `Server` or the `User-Agent` header. In an enumerating scenario, the bot retrieves a list of extensions/URIs from a specified location, then it probes them using INVITE, REGISTER or OPTIONS messages. OPTIONS enumerating is preferable since it is stealthier (it does not ring the phones, nor raise suspicions about the registration process). The bot has to match each request with the response it triggered based on the call-ID and/or transaction identifier. It analyzes the response to determine 1) if the dialed extension exists and if it is registered to the target, 2) exists and it is temporarily unavailable, or 3) doesn't exist at all. The interpretation of responses depends on the target's type and version. For example, if the target carries an `OpenSER sip proxy (version 1.1.1-notls)` fingerprint, the response to an OPTIONS message destined to an extension is interpreted as follows: a "200 OK" means that the extension exists and it is registered, a "404 NOT FOUND" means that the extension is invalid, but a received "100 TRYING" before a final error response means that the extension is valid but not available for the moment.

Cracking

Remote brute force password cracking consists in repeatedly trying guesses for an account's password using INVITE or REGISTER requests. Unchanged default passwords of deployed VoIP platforms make them strongly vulnerable to such attacks. In a cracking scenario, the bot has to discover the registration or the voicemail password for a user name. Note that the user name used in the authentication process is not always the same as the one in the user's URI (which can be obtained by enumerating). The attacker has to know the user name or the voice mail extension before going to a brute force attack. The bot retrieves a list of default passwords corresponding to the target platform. For each guess, it sends a first REGISTER (or INVITE) asking for a challenge. An error response from the target gives him back a nonce. The bot calls its encryption engine to build a new request containing credentials based on the challenge and the temporal nonce. The target's final response decides if the guess was right or not.

Fingerprinting

Remote fingerprinting allows the attacker to identify the type and the version of the SIP target platform. The simplest method consists on sending an OPTIONS message and extraction of the manufacturer string from its response. This process can be fooled if the manufacturer string was intentionally falsified. Smarter fingerprinting schemes as described in [?] or in [?] can be supported as well. In a fingerprinting scenario, the bot is asked to fingerprint one or range of SIP extensions. The bot has to send back its results to the attacker or -in order to not disclose him- to put them in a specified location where he can access them later.

Exploiting Specific Vulnerabilities

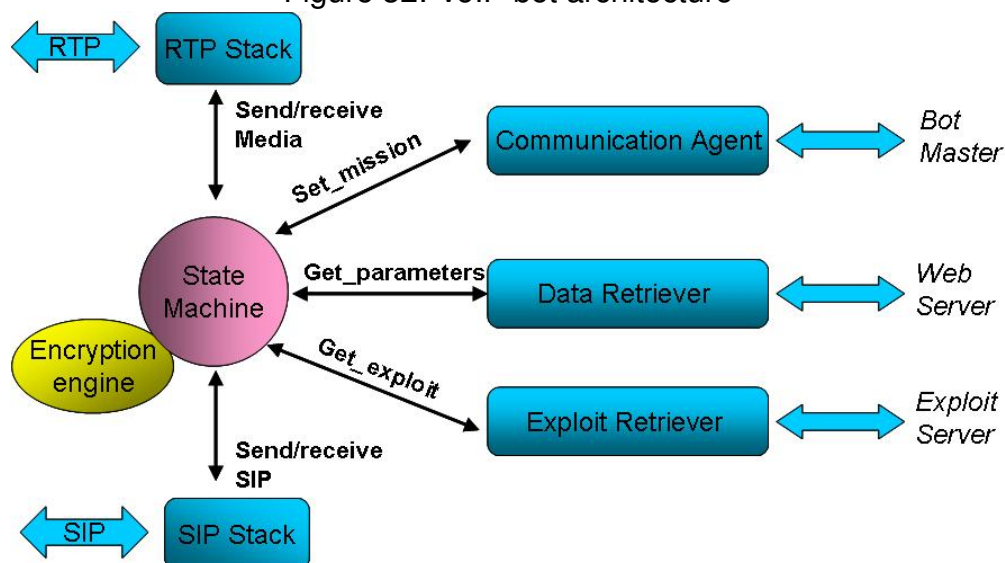
In an exploit attack scenario, the bot is either given the platform of the target or has to discover it by itself (by fingerprinting). The bot connects to an exploit server and retrieves a set of possible exploits corresponding to the target fingerprint. Each exploit should be tagged with some meta data so the bot can choose the exploit which meets the attacker's aim. In case of stateful attacks, the bot builds a local attack state machine to execute the attack given local and remote parameters as reported by the attacker.

10.2 Progress report

First we designed our bot architecture as shown in Figure 32. Its components are described next:

- The stack of different protocols: provides the bot with an application interface to use these protocols. The SIP stack is responsible for sending and receiving, manufacturing and parsing SIP messages. The RTP stack is responsible for coding and decoding, compressing and expanding, packetizing and depacketizing media flows.

Figure 32: VoIP bot architecture



Other stacks can be supported as well. For example, the STUN [?] protocol is useful to bypass NAT.

- The communication agent: allows the bot to exchange information and commands with the attacker. Most of the known botnets use IRC (Internet Relay Chat - RFC 1459) or peer-to-peer (P2P) networks for their control and command architecture. IRC is mainly designed for group communication and allows one-to-one communication (private discussion) as well. A channel (or a room) is supported by multiple servers building an application level spanning tree among them and relaying IRC messages between room visitors. P2P refers to a class of systems and applications that employ distributed resources to perform a function in a decentralized manner [?]. Bot masters moved towards P2P networks because of their high degree of anonymity and privacy. For sake of simplicity, we choose IRC as control and command infrastructure for our first prototype.
- The data retrieval component: allows the bot to retrieve different kinds of data (e.g. list of VoIP extensions (URLs), advertising audio files, list of default passwords to try, SIP messages to shoot, etc ...) using a data communication protocol (e.g. FTP or HTTP Client). Web servers using a dynamic DNS server (i.e. the DNS changes the IP corresponding to the web server over time) are preferred to avoid being tracked.
- The exploit retrieval component: allows the bot to retrieve specific exploits against vulnerabilities and software flaws in VoIP products. SIP is a strong candidate to become the UFBP (Universal Firewall Bypass Protocol) or the universal payload injector. This is assessed after the discovery of many cross-site scripting (XSS)⁴, SQL injection⁵, remote DoS on the target (similar to a ping of death)⁶ and remote eavesdropping⁷ exploits due to SIP vulnerabilities. These exploits are carried by mal-

⁴<http://seclists.org/fulldisclosure/2007/Oct/0174.html>

⁵http://voipsa.org/pipermail/voipsec_voipsa.org/2007-October/002466.html

⁶http://www.voipsa.org/pipermail/voipsec_voipsa.org/2007-August/002422.html

⁷http://www.voipsa.org/pipermail/voipsec_voipsa.org/2007-August/002424.html

formed SIP messages to attack SIP servers, embedded web servers and databases in the targeted systems. Some exploits are stateless (consisting on shooting one SIP message) but others are stateful (based on the state machine of the target). Stateful attacks are formed by a series of messages such as the content and the sending time of each message depends on the previous sent message and the corresponding reaction/response of the target to that message. The bot master defines a description syntax to describe stateful exploits and upload them to a server where they can be found by the bot. The bot parses the exploit description and builds a local state machine to perform the attack.

- The encryption engine: enables the bot to create digest authentication from credentials when authentication is required in the process of an attack or an attempt of registration. Typical use of this engine is password cracking and CPU-based flooding against the target authentication procedure.
- The SIP state machine: manages the operations of the bot with respect to the commands issued by the attacker. The mission of the bot as set by the attacker drives its behavior upon occurrence of SIP events (i.e. receiving a SIP request `RequestEvent` or a SIP response `ResponseEvent`) and `Timeout` events. The transition from a state to another is constrained by predicates on a set of global and local variables. For example, when receiving a 200 OK message that belongs to some existing dialog, the bot's next step is based on the `Cseq_method` (which determines the method the 200 OK is in response for) and on the global attack parameters (mission, target IP, target SIP port ...).

We implemented this architecture based on the IRC, SIP and RTP protocols using the Java language. For SIP we used the Jain SIP library [?]; for RTP we used the JMF library[?] and for IRC we used the PircBot library⁸. Our code is available under open source license at <http://gforge.inria.fr/projects/voipbot/>. The bot is currently able to perform DoS, SPIT, SCAN, CRAK, FINGERPRINT, SHOOT, EXPLOIT and REGISTER functionalities. Moreover, the bot master is able to perform a collective suicide of all the bots. The screenshot of Figure 33 shows the IRC client (Xchat⁹) of the bot master upon the connection of one bot.

Deployed on an Intel Pentium 4 CPU 3.40GHz and 2G RAM memory machine running a Linux kernel 2.6.18-1, the bot is able to send around 10,000 messages per second with different call-ids. The call-ID seed is the number of the bot as set by the attacker so messages from different bots have different Call-Ids. We used a similar machine with 3G RAM memory to be the target (hosting Asterisk and OpenSER¹⁰). Using legitimate messages and non existent URI destination, one bot is able to raise the target CPU to 100% in case of both Asterisk and OpenSER, and 2 bots are able to saturate the bandwidth of a LAN connection (about 12 MBytes/s). Asterisk consumes 25% of the host system memory (i.e. 750 MB) after 100 seconds of attack (i.e. 0.25% raise in memory/s), while OpenSER memory consumption depends on the number of its child processes as configured by the administrator (Each child reserves a 33 MB memory space).

⁸<http://www.jibble.org/javadocs/pircbot/index.html>

⁹<http://www.xchat.org/>

¹⁰The project has evolved in two parallel projects OpenSIPS and Kamailio

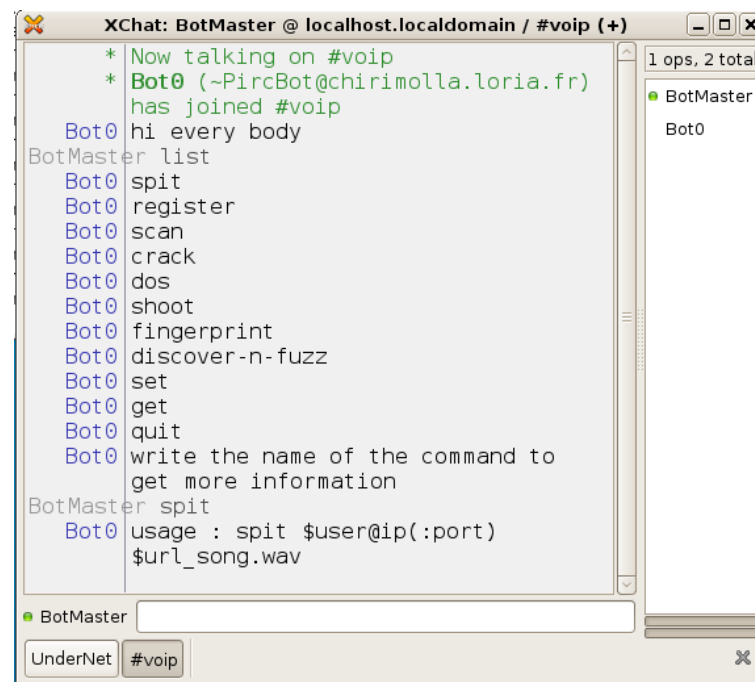


Figure 33: Screen view of the bot master

Open issues

We still have some implementation problems. For instance, the code is compiled and executed under JDK version "1.5.0_07" but it is not working with java version 1.6 (perhaps because some of the used libraries are not compatible with java1.6). The media tasks are handled as stand alone processes which can cause some portability issues. Instead, we prefer to handle them by threads using JMF or an alternative library. Currently, the bot supports only PCMU audio format because PCMA (which is the PSTN telephony norm in Europe) is not supported by JMF (For our knowledge). Introducing new audio formats in the code nominates it to the assessment of additional VoIP products like media gateways. All contributions, recommendations, extensions to the project are welcome.

10.3 Expected impact

This tool provides a framework for the assessment of VoIP domains, defense mechanisms, VoIP servers and equipments. The VoIP bots have proven high usefulness and efficiency when we used them in the evaluation of our specific intrusion detection models. VoIPbot is novel with respect to other VoIP security tools¹¹ because of its numerous functionalities and the ability it gives to manage distributed attacks. For example, by synchronizing and cooperating managed bots, we are able to bypass flooding defense mechanisms such as the PIKE defense module of OpenSER. Bots can be configured to emulate VoIP users with random or deterministic behaviors. In our experiments, we have used a set of 100 bot instances to generate attacks as well as to play the role of victim users. Several researchers, enterprises and the VoIPSA community have expressed their interest in this

¹¹<http://www.voipsa.org/Resources/tools.php>

tool from different perspectives. We expect that VoIPbot will attract much more attention with the emergence of discussions about future VoIP malwares.

11 LINUBIA

11.1 Presentation

The Internet is rapidly growing and fundamentally influencing economic, cultural, and social developments worldwide. While more and more people take advantage of this global network and new services are being deployed every day, more network resources are consumed. This creates the need for effective accounting mechanisms that are closely coupled to authentication mechanisms, e.g., in support of network management tasks, charging requirements, or intrusion detection systems for systems and users. Often it becomes necessary to know what amount and which type of network traffic a specific network user is generating. Today, as networking is moving towards an all-IP network [9] an accounting system integrated into the IP layer seems the most straight forward solution. This approach allows for the same accounting mechanisms to be used regardless of the application and the transport protocol carried over IP, or the data link layer and physical connection the IP runs on top of.

Although the accounting of IP network traffic has received wide attention since the beginning of the Internet [18], existing systems have a major drawback by looking strictly to the IP packet captured on the wire. Such an approach only allows for the mapping of each IP packet to the end-system which sent or received the packet, but it is unable to specify, which user was responsible for generating the traffic. Multi-user operating systems often use a single IP address, which is shared among different individual users. Since multiple users may be connected remotely at the same time to the same machine and may have different applications that generate IP traffic being transported over the network, it is impossible to identify how much traffic each of these users generated by just looking into the IP traffic at the router level.

Therefore, LINUBIA, uses a Linux kernel extension and a library for accessing this extension, for mapping each IP packet sent or received to the responsible user. This solution allows for splitting network costs in case of usage-based charging or may allow detection of the user or process that was responsible for illegal IP traffic.

Figure 11.3.2 depicts a typical scenario for using the new user-based IP accounting infrastructure. In an enterprise network users are typically authenticated by using a centralized authentication server such as LDAP (Light-weight Directory Access Protocol) [15] or Kerberos [12] and they may access the network from any terminal or working station that is configured to use the central authentication server. Upon authentication, the device to which the user logged on to starts to meter the network usage and sends periodic accounting records to the accounting server. Since the network usage is mapped to user identifiers (ID) and a user uses the same ID with any device he is allowed to connect to, the accounting server may aggregate the network usage from different devices within the network and present users with detailed and aggregated information about network traffic they generated.

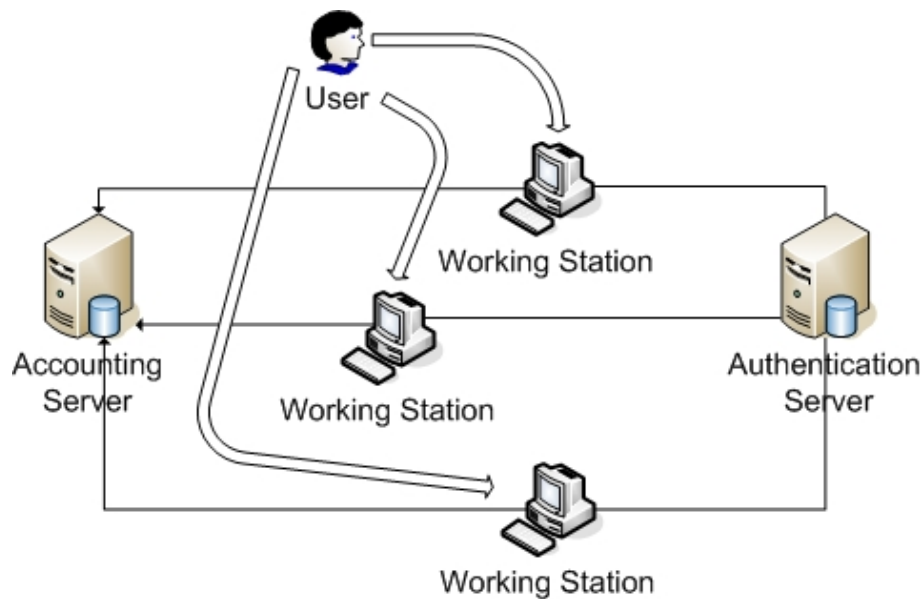


Figure 34: Enterprise Scenario

11.2 Motivating Use Cases

Based on the following use cases a summary of the main requirements for the new user-based IP accounting architecture, termed LINUBIA is derived. Based on these requirements that have been identified the design of the proposed solution is detailed.

11.2.1 Network Traffic Billing System

The first scenario deals with the case of a grid infrastructure spanning across a larger area on top of which customers may run their own grid applications. A grid user will typically install its applications on multiple nodes and these run typically with the users privileges. The grid operator may use the user-based accounting module in order to split network costs (traffic created by grid applications is typically high) among all customers based on the amount of traffic they created.

11.2.2 Individual Load Monitoring and Abuse Detection

The second scenario addresses the case of an institution, for example a university, which offers its students the possibility to use the Web for research and communication purposes, but does not want them to excessively waste precious network bandwidth for sharing videos, filesharing, and the like. The system setup is done in a way that a student can log into one of many computers at the university with his personal credentials. The user account information is stored in a centralized LDAP directory, so a specific student has uses the same user ID (UID) in every system he logs into. A script can regularly copy usage information to a database server, where it is stored and accumulated with the traffic

footprint of other users in order to detect possible anomalies in the traffic under investigation. The system administrator has the possibility to monitor network usage of students, independent of applications or the computer they use. With the help of this information he can detect and quantify abuses, suspend accounts of the respective users, or initiate further investigations.

11.2.3 Service Load Measurement

The third scenario handles the identification of applications, which generate abnormal traffic. For example, on a Linux server different services may be operational, some of them may not be using well-known ports (e.g., a bit-torrent client, which constantly changes ports it is running on). On that router connecting this server to the Internet, the administrator can monitor how much traffic this server created, but he can only identify applications based on port numbers. In case of applications that change these ports the use a user-based IP accounting module eases traffic monitoring for these type of applications.

11.3 Design and Implementation

11.3.1 Requirements

Based on these use cases described above, the following four key requirements for IP accounting LINUBIA have been identified:

- The IP accounting module shall account for IPv4 and IPv6 traffic information on a per-user bases operating the Linux operating system.
- The IP accounting module shall allow for application-based traffic accounting.
- An API interface shall be available for configuring the IP accounting module and retrieving accounted for data.
- The performance impact of the IP accounting module on the networking subsystem should be kept minimal.

11.3.2 LINUBIA Architecture

The architecture of an enterprise network having LINUBIA running on the Linux end-hosts, consists of both a network architecture (cf. Figure 11.3.2) that defines the network components required for LINUBIA and a end-host architecture (c.f. Figure 11.3.2) that defines the software components required within an end-system to support LINUBIA. Two types of devices may be identified: regular Linux hosts, which may be used by users and the accounting domain infrastructure (consisting of an authentication server, a data aggregation server, and a storage server). Linux nodes use an authentication server for verifying user credentials. Whenever a user logs in to a Linux host all the processes started by the user will run with the global UID of the user. Each Linux host has LINUBIA enabled. Accounted

for data is encapsulated in accounting records and it is transported from each Linux host to an accounting server using the Diameter protocol. The accounting server further stores the accounting records in a central database. For supporting this an accounting client runs on each host, collects the data accounted by LINUBIA and sends it to an accounting server using the Diameter protocol.

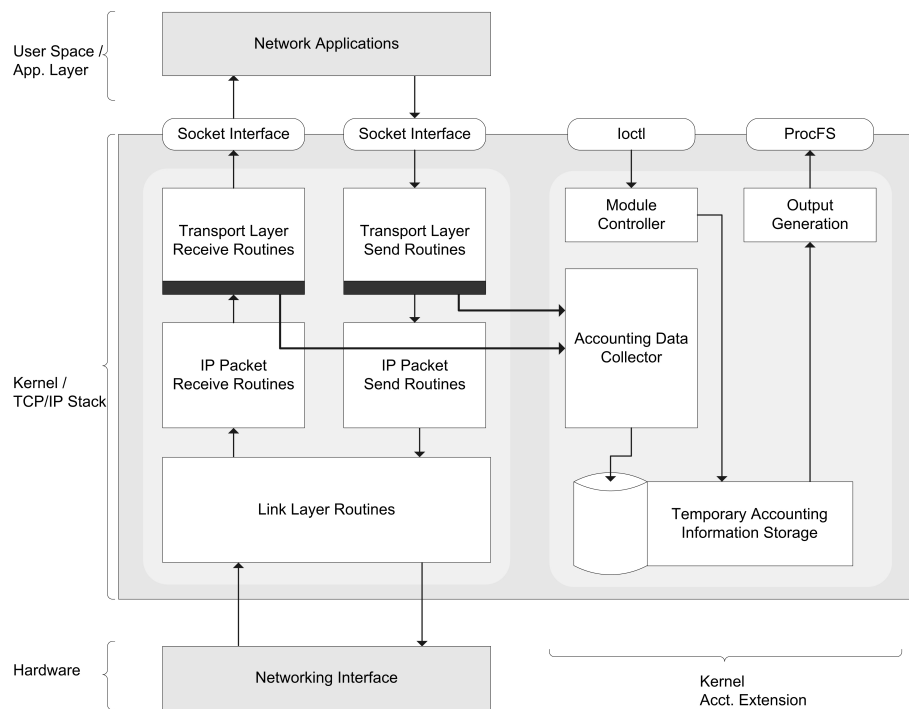


Figure 35: Linubia Architecture

11.3.3 End-Host Accounting Architecture

This section shall describe in detail the different components and their interactions required within an end-host in order to support user-based IP accounting. Regular operating systems do not offer a function to autonomously measure user-specific IP traffic. Therefore, a host needs to be modified in order to be able to perform such a task. Figure 11.3.2 shows how this can be achieved by modifying the Linux operating system kernel, which resides between the networking hardware and applications in the user space. The kernel allows network applications to access the TCP/IP stack via the network socket interface; it contains routines to send outgoing IP packets to the network and deliver incoming packets to the destination applications. These routines and the kernel have to be extended in order to measure, store and export the desired accounting information associated with each accounting-relevant IP network operation. This is done by a kernel accounting extension that consists of a number of components which are added to the kernel. The information storage component is responsible for the temporary storage of accounting information collected. The data collector component retrieves the necessary information from the IP networking subsystem and puts it in the storage component. The output generation component reformats the internal data before exporting it to user space via the proc filesystem

(procs). The module controller provides facilities to manage records stored, for example to reset all records of a specific user. It uses the ioctl interface. This architecture is designed to extract and export user-specific IP accounting information from the kernel to user space for further processing. The data is stored temporarily in the main memory by the kernel module. Data aggregation and persistent storage are done outside the kernel. in order to keep the load on the kernel as low as possible.

11.3.4 Integrated View

In addition to the kernel-based accounting architecture sketched in Section 11.3.3, two additional components are required for building accounting applications on top of LINUBIA. The first component is an accounting library that provides the API for querying and configuring the accounting module. It enables applications to access the kernel interfaces of the accounting extension. The second component is a Diameter accounting client that uses this library to fetch the user-based IP accounting records from the kernel and sends them to a remote data aggregation server using the Diameter protocol. The aggregation server can evaluate and store the accounting data persistently, for example by using a separate database server. A flexible system authentication back-end and Name Service Switch (NSS) configuration allows that a unique user account of a centralized user database (on a remote directory) can be used on any user host; the suggested interface being used for this is LDAP. The intention is that multiple hosts use the same user database and therefore the same UIDs for individual users, making users and associated accounting records uniquely identifiable across distinct hosts.

11.3.5 Implementation

The implementation of the host-based extension is based on the code layout of the useripacct project ?? and is entirely written in the C programming language. Compared to the other investigated approaches, LINUBIA supports 64 bit counters, provides real-time traffic statistics and allows parallel accounting of IPv4 as well as IPv6. The accounting system was implemented for modern 2.6 series Linux kernels and supports both IPv4 and IPv6. The information triplet to be extracted from each IP network operation consists of the IP packet size, the packet owner (user), and the network and transport protocols involved with the operation. Unfortunately, the required routines and protocol headers are distinct for IPv4 and IPv6, and for incoming traffic, the information cannot be retrieved at the IP layer, like it is the case for outgoing traffic. This required the embedding of the accounting module routines in the transport layer implementation. A shortcoming of this approach is a scatter of the LINUBIA code across several files in the Linux kernel network subsystem. The data collector can extract the size of a packet from IP packet headers; the sum of the transferred IP packet sizes equals the IP traffic. The network and transport protocol types can be determined by identifying the kind of the network routine or by also inspecting the IP packet header. The user information can be determined by looking up the ownership properties of the network socket corresponding to a packet. As it is possible that IP packets are sent or received that have no associated local network socket, there are rare situations where traffic cannot be attributed to a regular user. This is handled by

directing such accounting information to the record of a special user nobody. The information storage component is implemented as a number of records that are connected in groups of doubly-linked lists. Each record contains the UID as the primary identification attribute as well as the measured IP traffic values for different network and transport protocols. Users are dynamically added when they start using IP-based networking. Upon request, the output generation component loops through these lists to create a table with all users and their traffic records which is exported to the proc file system. The user space library reads a special item in the proc filesystem that is exported by the kernel extension and contains the temporary accounting information. The library recreates the record structures so that they can be easily accessed by other applications, such as the accounting client. It also provides functions to send commands to the module controller, using the ioctl interface. The accounting client sends locally detected accounting records to the accounting server using the Diameter protocol. Within Diameter, records are structured as sets of (predefined) Attribute-Value Pairs (AVP). The sample accounting client and sample server communicate in regular intervals by using accounting sessions, where an accounting session contains current records for one user, as delivered by the accounting library. Besides the accounting Attribute-Value-Pairs (AVPs) proposed in ??, a set of parameters have been defined as shown in Table 5.

Table 5: New AVPs for Linux User-Based IP Accounting

AVP Name	AVP Code	AVP Name	AVP Code
Linux-Input-IPV4-Octets	5001	Linux-Input-IPV4-TCP-Octets	5101
Linux-Output-IPV4-Octets	5002	Linux-Output-IPV4-TCP-Octets	5102
Linux-Input-IPV6-Octets	5003	Linux-Input-IPV4-UDP-Octets	5103
Linux-Output-IPV6-Octets	5004	Linux-Output-IPV4-UDP-Octets	5104
Linux-Input-TCP-Octets	5005	Linux-Input-IPV6-TCP-Octets	5105
Linux-Output-TCP-Octets	5006	Linux-Output-IPV6-TCP-Octets	5106
Linux-Input-UDP-Octets	5007	Linux-Input-IPV6-UDP-Octets	5107
Linux-Output-UDP-Octets	5008	Linux-Output-IPV6-UDP-Octets	5108

11.4 Evaluation

The evaluation of LINUBIA was performed both in terms of functional and performance evaluation. The tests have shown that the initial requirements have been fully met. The set of experiments that have been performed in order to test the functionality, accuracy and performance of the accounting module used a network set-up as the one described in Figure 36. The testing environment consists of two hosts that are connected in a LAN by a Fast Ethernet switch as seen in the figure. Both hosts run a Linux 2.6 operating system and use IPv4 as well as IPv6. Both hosts have Fast Ethernet network adapters. For testing the accuracy of the accounting module several tests have been performed in which TCP, UDP, and ICMP incoming and outgoing IPv4 and IPv6 traffic was generated and accounted for. The experiments have shown that the accounting module correctly accounts for IP traffic. During experiments it was observed that some traffic cannot be

mapped to any user (such as scanning traffic or incoming ICMP messages). Such traffic is accounted for the system user by the accounting module. Another observation concerns ICMP traffic that appears to be exclusively mapped to the system user and not to the user who actually sent the message. The reason for this is that raw socket operations are considered critical and only possible for user root, also for security reasons (a regular user can only execute the ping program because it has the SUID-bit set, thus being executed under root context).

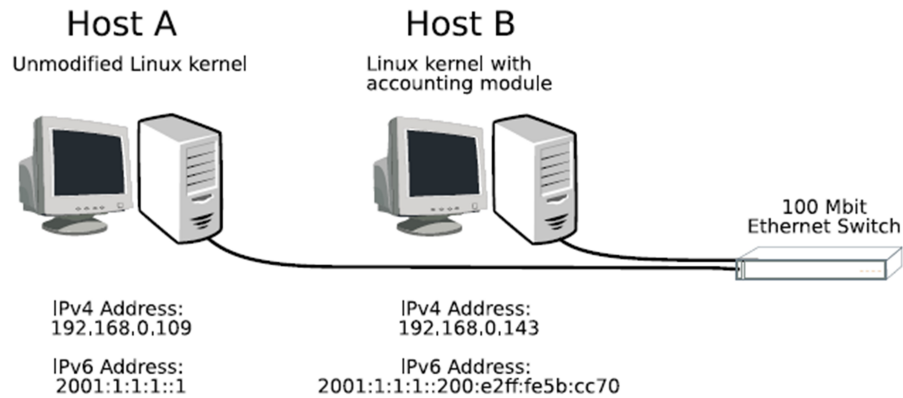


Figure 36: Evaluation Scenario

Table 6 shows the results of a first test consisting of a 256 MB file transfer over a Fast Ethernet link with and without LINUBIA using IPv4 and IPv6. The purpose of this test was to identify the impact of accounting on the performance of the Linux network subsystem. As the table shows there is only a small impact (0.83% for IPv4 and 0.41% for IPv6) on performance observed when running with LINUBIA enabled.

Table 6: Average time for a 256 MB file transfer over a Fast Ethernet connection with and without the user-based IP accounting enabled (average numbers of 20 runs)

	Unmodified IPv4	LINUBIA IPv4	Unmodified IPv6	LINUBIA IPv6
Avg. time	21.085 s	21.998 s	22.102 s	22.193 s
Std. deviation	0.062 s	0.208 s	0.010 s	0.204 s

Table 7: Average maximum throughput observed and calculated by Iperf over an Fast Ethernet connection, with and without the user-based IP accounting module enabled.

	Unmod. IPv4	LINUBIA IPv4	Rel. diff. (%)	Unmod. IPv4	LINUBIA IPv4	Rel. diff. (%)
Manual (Mbps)	93.880	93.099	0.839	92.661	92.281	0.412
Iperf (Mbps)	94.080	91.700	2.595	92.880	92.870	0.012

In Table 7 observed and estimated maximum throughput on a Linux box with and without LINUBIA are shown. For estimating the maximum throughput the Iperf ?? tool was used. The test with Iperf affirms that the measuring results are correct. Although the values are not totally equal, the dimensions are the same and the performance loss is marginal. During the evaluation phase of LINUBIA the architecture and its implementation have been tested to check the functionality they provide and the performance impact on the Linux kernel network subsystem. These tests have shown that LINUBIA delivers the required accounting results, especially a per-user network activities result on a multi-user operating system, while having a small impact on the performance of the end-system under investigation.

11.5 Expected Impact

Linubia was implemented as a prototype and is fully functional. Several grid-related projects (such as EC-GIN or Agrogrid) have expressed interest of using Linubia as a solution for accounting traffic created by different grid services. For achieving this goal, Linubia requires little change. As a future extension, exporting accounted data via the IPFIX protocol is foreseen. Such an approach could make LINUBIA even more interesting. Linubia is available under a GPL license. Current developers of Linubia are: Cristian Morariu(UZH) and Manuel Feier (UZH).

11.6 Progress Report

A prototype of Linubia is available for download at <http://www.csg.uzh.ch/staff/morariu/linubia/>. The software is available under a GPL licence.

11.7 Conclusion

LINUBIA demonstrates by a design and prototypical implementation that a user-based IP accounting approach is technically possible on modern Linux (2.6 series) operating systems. Additionally, it can be used also in the same version with the IPv6 protocol and it can be integrated into an existing accounting infrastructure, such as Diameter. On one hand, users are not supposed to have only one computer device of their own (not to mention sharing one device with other users), but rather to have several devices for different purposes. On the other hand, the more computers become commodities for daily life and will be used by different people (producing networking-related and other costs), the more important it becomes to establish accounting systems, which offer a clear and secure user identification on the end-device and will probably have an integrated character. The current implementation shows a clear proof of concept. Improvements are possible, e.g., with the storage component, which can be done with a smaller memory footprint and also more efficiently by utilizing advanced data structures that will help to optimize access times. Another interesting issue determines the linkage of the networking subsystem to the socket interface, which also implies a link to the process management of the operating system. An advanced accounting module can offer IP accounting not only per user, but

also per process. This allows for the identification, the management, or schedulability of processes not only by their CPU usage or memory consumption, but also by their network resource consumption. Finally, this may lead to the creation of network filters or firewalls that shall allow or deny network access to specific applications or users running on a host, instead of only allowing or denying specific services.

12 NetViz

NetViz provides a graphical visualization of security zones and virtual systems in Juniper NetScreen Security Systems. Systems of the Juniper NetScreen Security family offer an integrated solution comprising firewalls, VPN endpoints and traffic management systems. Due to an underlying virtualization concept, a high degree of flexibility as well as a simply customizable approach is achieved. By means of virtual LANs, virtual routers, and security zones, a complex network comprising various zones and corresponding security policies can be realized in a single system.

In order to visualize these complex structures in Juniper NetScreen Security Systems, NetViz has been developed and it allows for a graphical representation of coherences and interdependencies between virtual routers, virtual LANs, security zones, and virtual systems.

12.1 Presentation

NetViz reads the configuration of a Netscreen system and generates a graphic from it, it runs on machines with

- Perl version 5.8 or newer and modules
- Getopt::Long
- SVG

and it is available at <http://code.google.com/p/jp-parser/> under the Artistic License/GPLv2.

The output is either a SVG (Scalable Vector Graphics) or DOT file. To view the SVG file, a program to view vector graphics is needed, this can be done with most modern browser software. DOT is a powerful markup language for the graphical presentation of graphs, it describes the structure of directed or undirected graphs. To view a DOT file, a renderer is needed, e.g. *fdp* from the Graphviz package [?].

NetViz has been developed with Perl because Perl is platform-independent and very popular in server environments. In addition, it provides great possibilities to handle String with regular expressions. In a first approach, important elements had been extracted from the unstructured configuration file. This became unnecessary when virtual systems, router, zones and interfaces were represented as objects in Perl; the objects offer methods to generate the output. Each object draws itself and superior elements embed graphics of the objects they contain.

The first approach to output generation was the use of the DOT markup language. As the structural information from the configuration file could not be translated easily the resulting graphs were partly not clearly arranged. Thus, NetViz was changed to produce the graphs with ImageMagick¹² respectively PerlMagick¹³. Clearly structured graphs were

¹²<http://www.imagemagick.org>

¹³<http://www.imagemagick.org/script/perl-magick.php>

the result but the bitmap images did not scale well and to save storage compression was needed. The solution were vector graphics and SVG, and with the Perl extension SVG it was possible to generate valid SVG documents. This results in small files and the graphics scale very well.

NetViz is partitioned into three parts:

- **Parser**

The parser processes input data. This is divided into three parts, data input from the command line, the NetViz configuration file, and the configuration of the Netscreen system.

- **Wrapper**

The Wrapper calls the according procedures for the relevant lines of the Netscreen configuration. Commands are interpreted so that they can be passed to the objects from the data structure and to create objects.

- **Data**

The actual information about the Netscreen system is processed with four classes: `jp_router.pm`, `jp_vsys.pm`, `jp_zone.pm`, and `jp_interface.pm`. They represent all relevant objects in the Netscreen system and draw the output as each object generates its own graphical representation.

Details can be found in the NetViz man pages.

12.2 Impact

The proposed tool constitutes an entirely new form of graphical representation of complex structures in Juniper NetScreen systems that are currently only being reflected by means of text based configuration files. Moreover, as a result of the intended high degree of extensibility, the proposed tool can be easily adapted to the specific requirements of the EMANICS members.

12.3 Progress Report

NetViz started as a new development and was not based on an existing code basis. It starts from the configuration file of a Netscreen system. Such a file consists of a list of all commands, that have to be entered on the command line interface to achieve the desired configuration. Thus, it is sufficient to find those commands relating to routing and interpret them. In the following, we present an example with some simple zones:

```
set vrouter "untrust-vr" default-vrouter

set vrouter trust-vr
exit
```

```
set zone "Trust" vrouter "trust-vr"
set zone "Untrst" vrouter "untrust-vr"
set zone "DMZ" vrouter "trust-vr"
set zone "VLAN" vrouter "trust-vr"

set interface "ethernet2/1.8" tag 8 zone "DMZ"
set interface ethernet2/1.8 ip 10.10.66.254/24
set interface ethernet2/1.8 ip manageable
set interface ethernet2/1.8 ip manage ssh
set interface ethernet2/1.8 route

set vsys "VSYSINET" zone 19 vrouter name "inet-vr" id 1026
set vrouter "inet-vr" default-vrouter
set zone "Untrust" vrouter "inet-vr"
set interface id 128 "inet" zone "Untrust"
set interface ethernet4/1 group inet
set interface ethernet4/2 group inet
set interface ethernet3/1 zone "Untrust"
set interface ethernet3/1 ip 192.168.12.254/16
exit

set vrouter "untrust-vr"
set route 192.168.0.0/16 vrouter "inet-vr" preference 20 metric 1
set route 10.10.66.0/24 vrouter "trust-vr" preference 20 metric 1
exit

set vrouter "trust-vr"
set route 0.0.0.0/0 vrouter "untrust-vr" preference 20 metric 1
exit
```

Figure 37 shows the output produced directly as SVG and Figure 38 shows the output in DOT format rendered with fdp.

In the following we will explain the most important commands:

12.3.1 set vsys <name> zone <number> vrouter <VR-name> id <id-number>

The command creates a virtual system and defines the default router for this system, the routed gets an id. The part zone <number> is not part of the Juniper documentation and apparently has no effect but still it is contained in the configuration file of the Netscreen system.

12.3.2 set vrouter <name> <command>

With this command, the virtual router <name> is configured, if it does not yet exist it is created newly. The router can be defined as *sharable* meaning that it not only exists in the system where it was created but also in all contained systems.

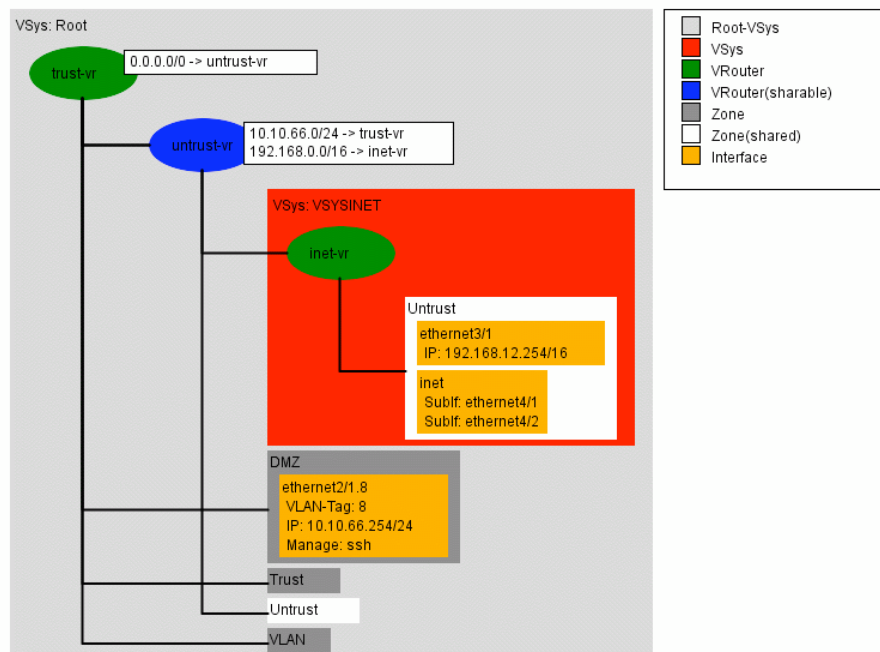


Figure 37: NetViz output as SVG

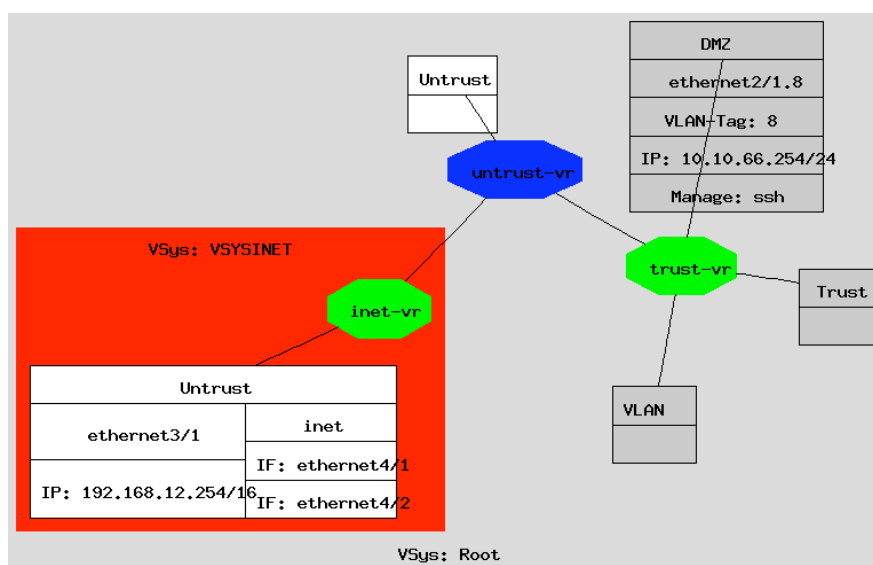


Figure 38: NetViz output in DOT format, rendered with fdp

If `set vrouter <name>` is called without further commands a virtual router is created. At the same time, a command block is opened by this. All following commands relate to this router, in this connection routes can be created for example with a command like

```
set route 137.193.0.0/16 vrouter 'untrust-vr \\'
preference 20 metric 1
```

The block is left with `exit`.

12.3.3 `set zone <name> <command>`

This command establishes a security zone, in a zone are interfaces. Security policies are defined for a zone and affect all interfaces contained. Additionally, default routers can be declared in a zone.

12.3.4 `set interface <name> <command>`

With this command, an interface is created, an interface is assigned to a zone. In general, an interface has an address space of IP addresses that are connected to this interface as well as a router that forwards the interface's traffic. An interface can be a physical interface, a physical interface can be divided into several interfaces with VLAN tags. Multiple interfaces can be combined to a group and such a group can then be used like a normal interface.

12.4 Conclusion

NetViz generates a graphical visualization from the configuration file of a Netscreen system. It outputs either an SVG file that can be displayed or a DOT file that has to be rendered. NetViz can be extended in many ways. First, a new output format can be added. This is rather simple, the new format has to be added to the parse options so that it is accepted as correct output format and one line has to be added to generate the output in the new format. E.g., to add jpg output a tool can be used that converts the SVG output to jpg.

Second, a new command of the CLI can be added. When a new command has to be added to the commands that are recognized it has to be added to the method `parse_file`. Then, the command has to be integrated into the module `jp_commands.pm` either by creating a new procedure or by including it into an existing procedure. Third, the SVG output can be designed interactive by using links. Then it would even be possible to send commands to the system and activate and deactivate interfaces by using the graphical representation. Also, via an SNMP interface the load and other information could be gathered and added to the graphics.

13 SMI to YANG Specification Translation

The NETMOD working group of the IETF is working on a data modeling language called YANG [21, 22] to be used with the NETCONF protocol published in RFC 4741 [9]. This section describes an extension of `libsmi` that translates SMIv2 [3, 4, 5] data models into YANG data models.

13.1 Presentation

The following sections outline the translation algorithm by showing the mapping of SMIv2 constructs to YANG constructs. The mapping is illustrated by considering an the IF-MIB [23] as an example SMIv2 module and its translation.

13.1.1 Translation of Special Types

The SMIv2 base types and some well known derived textual-conventions are mapped to YANG types according to the Table 8. The mapping of the `OCTET STRING` depends on the context. If an `OCTET STRING` type has an associated `DISPLAY-HINT`, then the corresponding YANG base type is the `string` type. Otherwise, the `binary` type is used. Similarly, the mapping of the `INTEGER` type depends on its usage as an enumeration or a 32-bit integral type.

SMIv2 Module	SMIv2 Type	YANG Module	YANG Type
SNMPv2-SMI	INTEGER		enumeration
SNMPv2-SMI	INTEGER		int32
SNMPv2-SMI	Integer32		int32
SNMPv2-SMI	OCTET STRING		binary
SNMPv2-SMI	OCTET STRING		string
SNMPv2-SMI	OBJECT IDENTIFIER	yang-types	object-identifier
SNMPv2-SMI	BITS		bits
SNMPv2-SMI	IpAddress	inet-types	ipv4-address
SNMPv2-SMI	Counter32	yang-types	counter32
SNMPv2-SMI	Gauge32	yang-types	gauge32
SNMPv2-SMI	TimeTicks	yang-types	timeticks
SNMPv2-SMI	Opaque		binary
SNMPv2-SMI	Counter64	yang-types	counter64
SNMPv2-SMI	Unsigned32		uint32
SNMPv2-TC	PhysAddress	yang-types	phys-address
SNMPv2-TC	MacAddress	ieee-types	mac-address
SNMPv2-TC	TimeStamp	yang-types	timestamp

Table 8: Mapping of SMIv2 base types and well known textual-conventions to YANG types

Note that the mappings shown in Table 8 may impact the imports of a module. Implementations must add any additional imports required by the mapping.

YANG / SMIv2 Module	Prefix
yang-types	yang
inet-types	inet
ieee-types	ieee
SNMPv2-TC	smiv2

Table 9: Special prefixes for well known SMIv2 and YANG modules

13.1.2 Module Prefix Generation Algorithm

The input to the prefix generation algorithm is a set of prefixes (usually derived from imported module names) and a specific module name to convert into a prefix. The algorithm described below produces a prefix for the given module name that is unique within the set of prefixes.

1. First, some fixed translations (see Table 9) mapping well known SMIv2 and YANG modules to short prefixes are tried. If a fixed translation rule exists and leads to a conflict free prefix, then the result of the fixed translation is used.
2. Otherwise, prefixes are generated by tokenizing an SMIv2 module name where hyphens are treated as token separators. The tokens associated with a module name are converted to lowercase characters. The shortest sequence of token concatenated using hyphens as separators which includes at least two tokens and is unique among all prefixes used in the set of prefixes associated with module names.

In the worst case, the prefix derived from an SMIv2 module name becomes the SMIv2 module name translated to lower-case. But on average, much shorter prefixes are generated.

13.1.3 Mapping of SMIv2 Modules

SMIv2 modules are mapped to a corresponding YANG modules. The YANG module name is the same as the SMIv2 module name.

The YANG namespace is constructed out of a constant prefix followed by the SMIv2 module name. Since SMIv2 module names are unique, the resulting YANG namespace is unique. The suggested prefix is `urn:ietf:params:xml:ns:yang:smiv2:` but usage of this prefix requires a specification and an associated IETF/IANA allocation action.

The YANG prefix is derived from the SMIv2 module name. Since the YANG prefix is supposed to be short and must be unique within the set of all prefixes used by a YANG module, the module prefix generation algorithm described in Section 13.1.2 is used.

The translation of the IF-MIB leads to the following YANG module frame. The prefix is the translation of the SMIv2 module name IF-MIB to lowercase (consisting of two token and thus no abbreviation).

SMIv2 Module	SMIv2 Symbol
SNMPv2-SMI	MODULE-IDENTITY
SNMPv2-SMI	OBJECT-IDENTITY
SNMPv2-SMI	OBJECT-TYPE
SNMPv2-SMI	NOTIFICATION-TYPE
SNMPv2-SMI	mib-2
SNMPv2-TC	TEXTUAL-CONVENTION
SNMPv2-CONF	OBJECT-GROUP
SNMPv2-CONF	NOTIFICATION-GROUP
SNMPv2-CONF	MODULE-COMPLIANCE
SNMPv2-CONF	AGENT-CAPABILITIES
SNMPv2-MIB	snmpTraps
SNMPv2-SMI	<i>all symbols</i>
SNMPv2-CONF	<i>all symbols</i>

Table 10: SMIv2 imports that are ignored in YANG

```

module IF-MIB {

    namespace "urn:ietf:params:xml:ns:yang:smiv2:IF-MIB";
    prefix "if-mib";

}

```

13.1.4 Mapping of SMIv2 Imports

SMIv2 `IMPORT` clauses are translated into YANG `import` statements. One major difference between the SMIv2 import mechanism and the YANG import mechanism is that SMIv2 imports specific symbols from a module while the YANG `import` statement imports all symbols of the referenced YANG module.

In order to produce correct and complete YANG `import` statements, it is necessary to apply the following rules:

1. Ignore all imports listed in Table 10. Note that the modules `SNMPv2-SMI` and `SNMPv2-CONF` are completely since all definitions in these modules are translated by translation rules.
2. Add any imports required by the type translations according to Table 8. This requires to consider all the types used in the translation unit.

The argument of the generated `import` statements are the untranslated SMIv2 module name. The `import` statement must contain a `prefix` statement. The prefixes are generated by applying the module prefix generation algorithm described in Section 13.1.2.

The translation of the `IMPORTS` of the `IF-MIB` leads to the following YANG `import` statements:

```
import IANAifType-MIB { prefix "ianaiftype-mib"; }
import SNMPv2-TC      { prefix "smiv2"; }
import yang-types      { prefix "yang"; }
```

13.1.5 Mapping of the MODULE-IDENTITY Macro

The clauses of the SMIv2 MODULE-IDENTITY macro are mapped to equivalent YANG statements.

- The SMIv2 ORGANIZATION clause is mapped to the YANG organization statement.
- The SMIv2 CONTACT-INFO clause is mapped to the YANG contact statement.
- The SMIv2 DESCRIPTION clause is mapped to the YANG description statement.
- Each SMIv2 REVISION clause is mapped to a YANG revision statement. The revision is identified by the date of contained in the SMIv2 REVISION. DESCRIPTION sub-clauses of REVISION clauses are mapped to corresponding description statement nested in revision clauses.
- The SMIv2 LAST-UPDATED is ignored if the associated date matches a REVISION clause. Otherwise, an additional revision statement is generated.
- The value of the invocation of an SMIv2 MODULE-IDENTITY macro is ignored.

The translation of the IF-MIB MODULE-IDENTITY macro invocation leads to the following YANG statements:

```
organization
  "IETF Interfaces MIB Working Group";

contact
  "Keith McCloghrie
  Cisco Systems, Inc.
  170 West Tasman Drive
  San Jose, CA 95134-1706
  US

  408-526-5260
  kzm@cisco.com";

description
  "The MIB module to describe generic objects for network
  interface sub-layers. This MIB is an updated version of
  MIB-II's ifTable, and incorporates the extensions defined in
  RFC 1229.";

revision "2000-06-14" {
```

```

    description
    "Clarifications agreed upon by the Interfaces MIB WG, and
    published as RFC 2863.";
}
revision "1996-02-28" {
    description
    "Revisions made by the Interfaces MIB WG, and published in
    RFC 2233.";
}
revision "1993-11-08" {
    description
    "Initial revision, published as part of RFC 1573.";
}

```

13.1.6 Mapping of the TEXTUAL-CONVENTION Macro

The SMIv2 uses invocations of the TEXTUAL-CONVENTION macro to define new types derived from the SMIv2 base types. Invocations of the TEXTUAL-CONVENTION macro are translated into YANG typedef statements.

The name of the TEXTUAL-CONVENTION macro invocation is used as the name of the generated typedef statement. The clauses of the SMIv2 TEXTUAL-CONVENTION macro are mapped to YANG statements embedded in the typedef statement as follows:

- The SMIv2 DISPLAY-HINT clause is used to determine the type mapping of types derived from the OCTET STRING type as explained in Section 13.1.1. Furthermore, the DISPLAY-HINT value is used to generate a regular expression for the YANG pattern statement within the type statement. The current implementation uses libsmi's DISPLAY-HINT to regular expression translation algorithm.
- The SMIv2 STATUS clause is mapped to the YANG status statement. The generation of the YANG status statement is skipped if the value of the STATUS clause is current.
- The SMIv2 DESCRIPTION clause is mapped to the YANG description statement.
- The SMIv2 REFERENCE clause is mapped to the YANG reference statement.
- The SMIv2 SYNTAX clause is mapped to the YANG type statement. SMIv2 range restrictions are mapped to YANG range statements while SMIv2 length restrictions are mapped to YANG length statements. SMIv2 INTEGER enumerations and SMIv2 BITS are mapped to YANG enum / value and bit / position statements.

The translation of the OwnerString and InterfaceIndex textual-conventions of the IF-MIB are shown below.

```

typedef OwnerString {
    type string {
        length "0..255";
    }
}

```

```

    pattern "\p{IsBasicLatin}{0,255}";
}
status deprecated;
description
  "This data type is used to model an administratively
  assigned name of the owner of a resource. This information
  is taken from the NVT ASCII character set. It is suggested
  that this name contain one or more of the following: ASCII
  form of the manager station's transport address, management
  station name (e.g., domain name), network management
  personnel's name, location, or phone number. In some cases
  the agent itself will be the owner of an entry. In these
  cases, this string shall be set to a string starting with
  'agent'.";
}

typedef InterfaceIndex {
  type int32 {
    range "1..2147483647";
  }
  description
    "A unique value, greater than zero, for each interface or
    interface sub-layer in the managed system. It is
    recommended that values are assigned contiguously starting
    from 1. The value for each interface sub-layer must remain
    constant at least from one re-initialization of the entity's
    network management system to the next re-initialization.";
}

```

13.1.7 Mapping of the OBJECT-TYPE Macro

The SMIv2 uses the OBJECT-TYPE macro to define objects and the structure of conceptual tables. Objects exist either as scalars (exactly one instance within an SNMP context) or columnar objects (zero or multiple instances within an SNMP context) within conceptual tables. A subset of columnar objects of a table define the index (key) of the table. Furthermore, conceptual tables can augment other conceptual tables. All these differences must be taken into account when mapping SMIv2 OBJECT-TYPE macro invocations to YANG.

Translating scalars and columnar objects: The SMIv2 OBJECT-TYPE macro invocations defining scalars or columnar objects are translated to YANG `leaf` statements. The name of the `leaf` is the name associated with the SMIv2 OBJECT-TYPE macro invocation.

1. The SMIv2 SYNTAX clause is mapped to the YANG `type` clause. Embedded clauses are generated as described in Section 13.1.6.
2. The SMIv2 UNITS clause is mapped to the YANG `units` statement.

3. The SMIv2 MAX-ACCESS clause is mapped to the YANG `config` statement with the value `false` if the value of the MAX-ACCESS clause is not `read-write` or `read-create`, or the context of the leaf requires it to be `false`.
4. The SMIv2 STATUS clause is mapped to the YANG `status` statement. The generation of the YANG `status` statement is skipped if the value of the STATUS clause is `current`.
5. The SMIv2 DESCRIPTION clause is mapped to the YANG `description` statement.
6. The SMIv2 REFERENCE clause is mapped to the YANG `reference` statement.
7. The value of the SMIv2 OBJECT-TYPE macro invocation is ignored.

Translating tree nodes and non-augmenting conceptual tables: In a first pass, iterate over the object identifier tree identifying all nodes that contain scalars and all non-augmenting conceptual tables. For each node, create a YANG `container` statement. For nodes representing groups of scalar objects, generate the necessary YANG `leaf` statements as described above. For nodes representing non-augmenting conceptual tables, identify the table entry OBJECT-TYPE, create a YANG `list` statement named after the SMIv2 entry OBJECT-TYPE. The rest of the clauses are translated as follows:

1. The SMIv2 SYNTAX clause is ignored.
2. The SMIv2 UNITS clause is ignored.
3. The SMIv2 MAX-ACCESS clause is ignored.
4. The SMIv2 STATUS clause is mapped to the YANG `status` statement. The generation of the YANG `status` statement is skipped if the value of the STATUS clause is `current`.
5. The SMIv2 DESCRIPTION clause is mapped to the YANG `description` statement.
6. The SMIv2 REFERENCE clause is mapped to the YANG `reference` statement.
7. The SMIv2 INDEX clause is mapped to the YANG `key` clause listing the columnar objects forming the key of the YANG list.
8. The value of the SMIv2 OBJECT-TYPE macro invocation is ignored.

Note that the SMIv2 non-augmenting conceptual table node is not translated to YANG. Within the `list` statement, create YANG `leaf` nodes as described above. For objects listed in the SMIv2 INDEX clause that are not part of the conceptual table itself, create YANG `leaf` statements of type `keyref` pointing to the referenced definition.

Translating augmenting conceptual tables: In a second pass, iterate over all augmenting conceptual tables. For each augmenting conceptual table, identify the table entry OBJECT-TYPE, create a YANG `augment` statement with the first argument containing the path of the augmented table. The rest of the clauses are translated as follows:

1. The SMIv2 SYNTAX clause is ignored.
2. The SMIv2 UNITS clause is ignored.
3. The SMIv2 MAX-ACCESS clause is ignored.
4. The SMIv2 STATUS clause is mapped to the YANG `status` statement. The generation of the YANG `status` statement is skipped if the value of the STATUS clause is `current`.
5. The SMIv2 DESCRIPTION clause is mapped to the YANG `description` statement.
6. The SMIv2 REFERENCE clause is mapped to the YANG `reference` statement.
7. The value of the SMIv2 OBJECT-TYPE macro invocation is ignored.

Note that the SMIv2 augmenting conceptual table node is not translated to YANG. Within the `augment` statement, create YANG `leaf` nodes as described above.

The translation of the some key parts of the IF-MIB are shown below.

```
container interfaces {

  leaf ifNumber {
    type int32;
    config false;
    description
      "The number of network interfaces (regardless of their
       current state) present on this system.";
  }

  list ifEntry {

    key "ifIndex";
    description
      "An entry containing management information applicable to a
       particular interface.";

    leaf ifIndex {
      type if-mib:InterfaceIndex;
      config false;
      description
        "A unique value, greater than zero, for each interface. It
         is recommended that values are assigned contiguously
         starting from 1. The value for each interface sub-layer
```

```
        must remain constant at least from one re-initialization of
        the entity's network management system to the next re-
        initialization.";
    }

    leaf ifDescr {
        type smiv2:DisplayString {
            length "0..255";
        }
        config false;
        description
            "A textual string containing information about the
            interface. This string should include the name of the
            manufacturer, the product name and the version of the
            interface hardware/software.";
    }

    /* skipping several definitions */

    leaf ifAdminStatus {
        type enumeration {
            enum up      { value 1; }
            enum down    { value 2; }
            enum testing { value 3; }
        }
        config true;
        description
            "The desired state of the interface. The testing(3) state
            indicates that no operational packets can be passed. When a
            managed system initializes, all interfaces start with
            ifAdminStatus in the down(2) state. As a result of either
            explicit management action or per configuration information
            retained by the managed system, ifAdminStatus is then
            changed to either the up(1) or testing(3) states (or remains
            in the down(2) state).";
    }

    /* skipping more definitions */
}

}

augment "/if-mib:interfaces/if-mib:ifEntry" {
    description
        "An entry containing additional management information
        applicable to a particular interface.";
```

```
leaf ifName {
    type smiv2:DisplayString;
    config false;
    description
        "The textual name of the interface. The value of this
        object should be the name of the interface as assigned by
        the local device and should be suitable for use in commands
        entered at the device's 'console'. This might be a text
        name, such as 'le0' or a simple port number, such as '1',
        depending on the interface naming syntax of the device. If
        several entries in the ifTable together represent a single
        interface as named by the device, then each will have the
        same value of ifName. Note that for an agent which responds
        to SNMP queries concerning an interface on some other
        (proxied) device, then the value of ifName for such an
        interface is the proxied device's local name for it.

        If there is no local name, or this object is otherwise not
        applicable, then this object contains a zero-length string.";
}

/* skipping more definitions */
}
```

13.1.8 Mapping of the NOTIFICATION-TYPE Macro

The SMIv2 provides the NOTIFICATION-TYPE macro to define notifications. YANG provides the `notification` statement for the same purpose. The name of the NOTIFICATION-TYPE macro invocation is used as the name of the generated `notification` statement. The clauses of the NOTIFICATION-TYPE macro are mapped to YANG statements embedded in the `notification` statement as follows.

- The SMIv2 OBJECTS clause is mapped to a sequence of YANG containers. For each object listed in the OBJECTS clause value, a YANG container statement is generated. The name of this container is the name of the notification and the name of the current concatenated by a hyphen. If the current object belongs a conceptual table, then a sequence of leaf statements is generated for each INDEX of the SMIv2 conceptual table. Next, a leaf statement is generated for the current object. All container leafs are marked as `config false`.
- The SMIv2 STATUS clause is mapped to the YANG `status` statement. The generation of the YANG `status` statement is skipped if the value of the STATUS clause is `current`.
- The SMIv2 DESCRIPTION clause is mapped to the YANG `description` statement.
- The SMIv2 REFERENCE clause is mapped to the YANG `reference` statement.
- The value of the SMIv2 NOTIFICATION-TYPE macro invocation is ignored.

The translation of the linkDown notification of the IF-MIB is shown below.

```
notification linkDown {
  description
    "A linkDown trap signifies that the SNMP entity, acting in
    an agent role, has detected that the ifOperStatus object for
    one of its communication links is about to enter the down
    state from some other state (but not from the notPresent
    state). This other state is indicated by the included value
    of ifOperStatus.";

  container linkDown-ifIndex {
    leaf ifIndex {
      type keyref {
        path "/if-mib:interfaces/if-mib:ifEntry/if-mib:ifIndex";
      }
      config false;
      description
        "Automagically generated keyref leaf.";
    }
  }

  container linkDown-ifAdminStatus {
    leaf ifIndex {
      type keyref {
        path "/if-mib:interfaces/if-mib:ifEntry/if-mib:ifIndex";
      }
      config false;
      description
        "Automagically generated keyref leaf.";
    }
    leaf ifAdminStatus {
      type enumeration {
        enum up      { value 1; }
        enum down    { value 2; }
        enum testing { value 3; }
      }
      config false;
      description
        "The desired state of the interface. The testing(3) state
        indicates that no operational packets can be passed. When a
        managed system initializes, all interfaces start with
        ifAdminStatus in the down(2) state. As a result of either
        explicit management action or per configuration information
        retained by the managed system, ifAdminStatus is then
        changed to either the up(1) or testing(3) states (or remains
        in the down(2) state).";
    }
  }
}
```

```

    }
  }

  container linkDown-ifOperStatus {
    leaf ifIndex {
      type keyref {
        path "/if-mib:interfaces/if-mib:ifEntry/if-mib:ifIndex";
      }
      config false;
      description
        "Automagically generated keyref leaf.";
    }
    leaf ifOperStatus {
      type enumeration {
        enum up { value 1; }
        enum down { value 2; }
        enum testing { value 3; }
        enum unknown { value 4; }
        enum dormant { value 5; }
        enum notPresent { value 6; }
        enum lowerLayerDown { value 7; }
      }
      config false;
      description
        "The current operational state of the interface. The
        testing(3) state indicates that no operational packets can
        be passed. If ifAdminStatus is down(2) then ifOperStatus
        should be down(2). If ifAdminStatus is changed to up(1)
        then ifOperStatus should change to up(1) if the interface is
        ready to transmit and receive network traffic; it should
        change to dormant(5) if the interface is waiting for
        external actions (such as a serial line waiting for an
        incoming connection); it should remain in the down(2) state
        if and only if there is a fault that prevents it from going
        to the up(1) state; it should remain in the notPresent(6)
        state if the interface has missing (typically, hardware)
        components.";
    }
  }
}

```

13.2 Expected Impact

The `libsmi` serves as an open source reference implementation of the SMIv1, SMIv2, SPPI, and SMIng network management data modeling languages. It is regularly used by

independent organizations like the IETF and many companies for quality testing of SMI MIB modules. Some companies maintain proprietary code generation back-ends for the `smidump` tool, which is part of the `libsmi` source package.

The NETMOD working group of the IETF is currently not chartered to work on coexistence issues and in particular SMIv2 to YANG specification translations. However, many working group participants seem to agree that automated translations of SMIv2 modules to YANG modules are necessary in order to provide access to the existing SMIv2 instrumentation available on many devices. The work described in this section is a step to fill this niche and to develop the translation that might be considered for standardization at some later point in time by the IETF.

13.3 Progress Report

The current implementation consists of a compiler backend that does conversions on the fly while generating output. This was simple to implement but has the drawback that there is no proper in memory representation of the translated YANG module. It would be desirable to have proper YANG data structures supported by the library plus a proper YANG pretty printer so that the conversion algorithm can simply fill in the required YANG data structures, leaving the pretty printing to a generic YANG pretty printer.

While most of the SMIv2 constructs are properly translated, there is currently no support for translations of `OBJECT-IDENTITY` macro invocations. A proper solution requires proper YANG language support (e.g., the `identity` statement), which is being discussed in the NETMOD working group of the IETF.

Furthermore, the translator ignores any SMIv2 conformance definitions. With the addition of the `feature` statement to YANG, it might be possible to translate SMIv2 conformance definitions into YANG `features`. But this requires more study once the YANG feature mechanism has been properly worked out.

The current mapping of SMIv2 notification definitions to YANG is very verbose and duplicates definitions. The IETF working group is discussing changes to YANG (`leafref`) that will make the conversion simpler.

Finally, the pattern generation algorithm needs more work. While the current algorithm seems to produce correct regular expressions for the more frequent cases, it does produce erroneous results in some corner cases. Furthermore, the generated regular expressions can be very long and difficult to read. More work is needed to define a translation algorithm that always produces correct results and tries to minimize the complexity of the generated regular expressions.

13.4 Conclusions

The `libsmi` open source C library for processing SMIv2 modules has been extended with a translator generating YANG modules from SMIv2 modules. While the current library does not yet cover all macros of SMIv2, it produces reasonable output for the most important constructs. The output produced by the translation algorithm has been verified for syntactic correctness using `pyang` on the generated modules.

If possible, future work should address the limitations spelled out in Section 13.3. In addition, it is desirable to produce a proper specification of the translation process in the form of an Internet-Draft in order to seek for proper review by the IETF experts and to motivate other SMIv2 tool makers to implement the same translation process in their software.

14 TomP2P: A Distributed Multi Map

Peer-to-peer (P2P) systems have several advantages over centralized systems, including load balancing, robustness, scalability, and fault tolerance. Many popular P2P libraries implement a distributed hash table with `get()` and `put()` operations. This section presents TomP2P, a Java-based P2P network, which implements a distributed multi map. This library includes besides `get()` and `put()`, also `add()` operations and further extensions. The API of TomP2P is described and future work is identified.

14.1 Introduction

Peer-to-peer (P2P) systems have several advantages over centralized systems, including load balancing, robustness, scalability, and fault tolerance. There are many P2P network implementations, which are used to address those issues, such as FreePastry [24], CAN [25], Bamboo [?], and Chord [26].

The basic of every P2P network is that it consists of equal peers that are interconnected. In contrast to client-server architecture, every peer in a P2P system can be a server and a client at the same time.

There are three types of P2P networks: structured, semi-structured, and unstructured. In the early stages of P2P network development and research, Napster and Gnutella determined the first and second generation of P2P networks. The former showed an approach with central elements representing a single point of failure. The latter was unstructured and did not scale very well. An introduction of super peers made the unstructured network semi-structured [27]. A super peer has higher capabilities in terms of resources or network properties. The most recent research focuses on structured networks. Structured networks such as distributed hash tables (DHTs), have usually a resource lookup complexity of $O(\log n)$, where n is the number of peers in the system.

TomP2P is a Java-based P2P network that is based on the XOR distance metric, similar to Kademlia [28]. The peer IDs are 160bit and routing [29] is iterative. TomP2P is available for download [30].

14.2 Distributed Hash Table

A distributed hash table implements basic operations `put` and `get`. Both operations require first to search (route) for the responsible peers. This routing operation depends on the distance metric that specifies the distance of two peers. Usually, the input for measuring distance is based on the ID of a peer. After the routing process, peers that are close, according to the distance metric, are queried either to store (`put`) or to retrieve (`get`) data. The routing process stops if no further close neighbors are found.

14.2.1 Fast Get

For `get` operations of existing data, the routing process can be shortened using *fast get*. Each queried peer replies if it contains the data. Depending on the replication factor, the

routing process stops if enough peers are found to retrieve the data.

- void put(key, value)
- value get(key)

TomP2P implements a distributed hash table with get and put with *fast get*. Furthermore, a *fast get* reply indicates the length of the data that can be retrieved, which allows TomP2P to choose UDP for short messages and TCP for long messages. In case of updates during and after a routing process, *i.e.*, if a peer gets outdated length of data to use UDP, an error message is sent back indicating to use TCP.

14.2.2 Reversed Conditional Get

Conditional get is used to avoid sending data that is present in local caches. For HTTP, conditional get is described in RFC 2616 [31]. A server replies with a time and etag, which changes if content changes. The client stores this time and etag with the data in its cache. If a client requests pages with an etag, the server checks if content has been modified. If a page has not been changed the server replies with “304 not modified”. However, this only works in case of one server. In P2P networks, due to redundancy, multiple peers (servers) can store data. Thus, the server cannot generate an etag that changes if data changes as it requires coordination with other peers. In P2P environment, the client that requests data need to create the etag. This etag is stored per requesting peer (client) on the replying peer (server). If a client requests data, the etag is sent, and if a server already replied to this conditional value and the value did not change, the reply message is a “not modified message”.

TomP2P implements reversed conditional get to reduce traffic. The etag are random values stored in a cache with size 1000 and LRU replacement strategy.

14.3 Distributed Hash Table Extensions

Distributed hash tables offer a simple and powerful way to store and retrieve data. However, it is often the case, that DHT operations in an application are used from different context. To avoid key collisions for same keys in different context, domains are introduced. Furthermore, in a distributed hash table, peers may fail at any time. Data that is never removed due to peer failures may cause memory leaks. Thus, old data need to be removed in a garbage collection process.

14.3.1 Domains

Domains are used to avoid key collisions, e.g., put(0x123, path, value1) that stores value1 under the key 0x123 put (0x123, “attribute”, value2) that stores value2 under the same key do not collide and can be retrieved using get(0x123, “path”), respectively get(0x123, “attribute”). Additional operations that are associated with the domain are move and copy.

The move operation moves data from one domain to another, while the copy operation copies data from one domain to another. Both operations move and copy data within a peer.

- void put(key, domain, value)
- value get(key, domain)
- void move(key, domain_old, domain_new)
- void copy(key, domain_old, domain_new)

TomP2P implements support for domains. For backwards compatibility, the domain can be set to a default.

14.3.2 Garbage Collection

Each entry has a time-to-live (TTL). This value indicates how long the data is valid until it needs to be renewed. After a TTL, data may be removed if storage space is scarce. In a cooperative environment, the requesting peer can specify the TTL, in a hostile environment, the TTL needs to be set from the receiving peer.

- void put(key, domain, value, ttl)
- void remove(key, domain)

TomP2P implements the TTL for a cooperative environment and has hooks that allows TomP2P set override the TTL on the receiving side. Furthermore, TomP2P has a remove method for removing data.

14.3.3 Distributed Multi Map

A multi map stores more than one value for a key. These values are stored in a collection. A hash table can be implemented with a multi map if the collection size is restricted to one.

An example where a multi map can be used are trackers in a file sharing application. The drawback of using a put operation is that adding data requires first to retrieve the values with a get operation, then values are updated and sent back using the put operation. This is inefficient for large trackers, as redundant data is sent over the network and routing is done twice. A multi map that uses an add operation reduces routing and redundant data.

- void add(key, domain, value, ttl)
- void remove(key, domain, value)
- int size(key, domain)

TomP2P implements add, remove, and the size operation. The multi map is stored chronologically in a set. Thus, duplicates are not allowed. In case of concurrent adds and retrieval, UDP may be chosen instead of TCP. In case of UDP, where the data has grown over the UDP limit due to concurrent updates, a “partially ok” message is sent, and the UDP packet is filled up to its limit.

14.4 Future Work

Future work includes a notification service. If a peer shuts down ordinarily, close neighbors shall be notified, in order to remove this peer from the routing process, or do data synchronization. Currently, data synchronization is not implemented. Such a notification and synchronization operations could be implemented using the following method.

- void notifyP2P(NodeAddress addr1, Action action)
- void sync(NodeAddress addr1, NodeAddress addr2, boolean master)

In hostile environments, public key need to be used to create a secure peer ID and to sign data in order to prevent others to overwrite data. As a consequence, operations that affect multiple values are put, add and remove operations. Thus, to support this feature, the following configuration method is required.

- void setMessageSignature(boolean signature)

For garbage collection, old data needs to be removed. Also if a peer restarts, data need to be checked if still valid. Currently, this works for the memory based multi map as a restart removes all entries. For the disk based, a list need to be stored that is always in sync with other content. An other possibility would be to check all data on startup, but this has proven to take a long time. In combination with garbage collection, the renew of date needs to be supported. Thus, the following method will be implement. Single entries can be specified using the hash value. However, two different values may have the same hash value.

- void refresh(key, domain)
- void refresh(key, domain, int hash)

A further extension is to allow computations. In a multi map a peer could store data and submit tasks that manipulate this data. Thus, to support this feature, the following configuration method is required. Redundancy is handled the same way as the other operations.

- void submitTask(key, domain, task)

15 SNMPDump packaging and distribution

15.1 Presentation

The `snmpdump` software tool, developed by the research group at Jacobs University in collaboration with the group at the University of Twente, converts `pcap` files containing SNMP traces to intermediate formats defined in RFC 5345 [32]. The tool supports

- reassembly of UDP datagrams,
- filtering of SNMP message elements,
- anonymization of SNMP message elements,
- conversion to XML and CSV formats, and,
- extraction of flows.

and serves as a reference implementation for RFC 5345 [32]. A description of the architecture of the tool has been published at IM 2007 [33] and a detailed discussion of SNMP trace analysis definitions has been published at AIMS 2008 [34].

The implementation of `snmpdump` depends on several libraries for reading `pcap` files, for the reassembly of IP packets, for cryptographic functions, and for accessing SMIPv2 definitions. While the `snmpdump` source code uses standard Unix software configuration management tools (`automake`, `autoconf`, `libtool`), it has been reported that operators participating in trace collection activities sometimes have trouble to get the tool compiled and installed, mainly due to the library dependencies and related versioning issues. To make it easier to install `snmpdump`, it was decided to package it for the Debian software distribution.

15.2 Expected Impact

The `snmpdump` tool is targeting a small user group, namely people interested in analyzing SNMP exchanges on a network or network operators cooperating with people interested in analyzing SNMP traffic. As such, the impact of packaging this tool cannot be measured using normal download statistics. It should be noted, however, that `snmpdump` is used by EMANICS WP2 to capture, process, and share SNMP traces and by EMANICS WP7 for the analysis of these traces. As such, this packaging activity has project internal impact.

WP2 members often collaborate with network operators to obtain traces from production networks and the need for an easy to install package of the `snmpdump` tool became apparent in these activities since people running production networks do not have the time to resolve library versioning issues in order to help research projects.

15.3 Progress Report

The original `snmpdump` source tree contained the source code of a reusable library of anonymization transforms (`libanon`) and the source code of the `snmpdump` application itself. Since `libanon` was designed to be reusable in other contexts, it was decided to package the `libanon` library separately. In order to avoid dependencies of the `libanon` Debian packages on the `snmpdump` source tree, it was decided to split the original `snmpdump` source tree by moving all the `libanon` files to a new source `libanon` source code repository and adding the standard Unix software configuration management scripts for `automake`, `autoconf`, and `libtool`.

In a second step, Debian packaging files were created for the `libanon` library. The resulting Debian packages for Intel platforms are `libanon0.0.1.0-1_i386.deb` (the runtime library) and `libanon-dev_0.0.1.0-1_i386.deb` (the development library). Standard Debian packaging tools and procedures were followed.

The third step was the packaging of `snmpdump` itself, resulting in the Debian Intel package `snmpdump_0.4.6-1_i386.deb` package. All Intel packages and the Debian packaging files are available from <http://www.eecs.jacobs-university.de/users/schoenw/debian/>.

The source code of `snmpdump`¹⁴ and `libanon`¹⁵ is hosted on two Trac instances. Trac¹⁶ is an enhanced wiki and issue tracking system for software development projects providing a direct interface to the subversion revision control system.

The `libanon` source code is distributed under the GNU Lesser General Public License and the packaging is licensed under the GNU GPL. The `snmpdump` source code is distributed under a BSD style license while the packaging is licensed under the GNU GPL.

15.4 Conclusion

The Debian packaging of `snmpdump` has been achieved by splitting the original source tree into a new source tree for the anonymization library and subsequent packaging into three Debian packages. A wiki and bug tracking environment is provided using Trac.

¹⁴<https://trac.eecs.iu-bremen.de/projects/snmpdump/>

¹⁵<https://trac.eecs.iu-bremen.de/projects/libanon/>

¹⁶<http://trac.edgewall.org/>

16 Weathermap packaging and distribution

16.1 Presentation

up to 3 pages presentation of the software package!

16.2 Expected Impact

0.25 to 1 page description of the impact expected by providing the packaged software to the community and provide as much data as possible to assess the impact, e.g. increase in number of downloads, increase in the contributing community, ...

16.3 Progress Report

1-2 pages of description of changes made with the support of EMANICS, provide the necessary data so as to allow any person to download the software, provide details about under which Open Source license the software is distributed, ...

16.4 Conclusion

17 Conclusions

Work-package 6 did, in phase 2 of the network of excellence, extend its activity with a third initiative dedicated to existing Open Source software packaging and tutoring. Dedicated software development/enhancement and inventory activities have been followed as well. As a result of the first call, seven activities among which two in packaging and one in inventory extension have been supported by work-package 6. 6 of these seven activities have been completed and related software packages, web-pages, distributions are all public.

A new call was issued in april 2008. The supported initiatives were reported in this release of the deliverable. A nal call covering the 4th year of EMANICS was issued in early december 2008. The selection process will be complete in early january 2009, enabling imediate start of the activities.

18 Abbreviations

19 Acknowledgement

This deliverable was made possible due to the large and open help of the WP6 Partners of the EMANICS NoE. Many thanks to all of them.

References

- [1] F. Strauß and J. Schönwälder. SMIng - Next Generation Structure of Management Information. RFC 3780, TU Braunschweig, IU Bremen, May 2004.
- [2] M. Rose and K. McCloghrie. Concise MIB Definitions. RFC 1212, Performance Systems International, Hughes LAN Systems, March 1991.
- [3] K. McCloghrie, D. Perkins, and J. Schönwälder. Structure of Management Information Version 2 (SMIv2). RFC 2578, Cisco Systems, SNMPinfo, TU Braunschweig, April 1999.
- [4] K. McCloghrie, D. Perkins, and J. Schönwälder. Textual Conventions for SMIv2. RFC 2579, Cisco Systems, SNMPinfo, TU Braunschweig, April 1999.
- [5] K. McCloghrie, D. Perkins, and J. Schönwälder. Conformance Statements for SMIv2. RFC 2580, Cisco Systems, SNMPinfo, TU Braunschweig, April 1999.
- [6] J. Schönwälder and F. Strauß. Next Generation Structure of Management Information for the Internet. In *Proc. 10th IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, number 1700 in LNCS, pages 93–106. Springer, October 1999.
- [7] J. Case, R. Mundy, D. Partain, and B. Stewart. Introduction and Applicability Statements for Internet Standard Management Framework. RFC 3410, SNMP Research, Network Associates Laboratories, Ericsson, December 2002.
- [8] K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, and A. Smith. COPS Usage for Policy Provisioning (COPS-PR). RFC 3084, Nortel Networks, Intel, Cisco, IPHighway, PFN, Allegro Networks, March 2001.
- [9] R. Enns. NETCONF Configuration Protocol. RFC 4741, Juniper Networks, December 2006.
- [10] F. Strauß and J. Schönwälder. Next Generation Structure of Management Information (SMIng) Mappings to the Simple Network Management Protocol (SNMP). RFC 3781, TU Braunschweig, IU Bremen, May 2004.
- [11] D. Harrington, R. Presuhn, and B. Wijnen. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. RFC 3411, Enterasys Networks, BMC Software, Lucent Technologies, December 2002.

- [12] D. Harrington and J. Schönwälder. Transport Subsystem for the Simple Network Management Protocol (SNMP). Internet Draft (work in progress) <draft-ietf-isms-tmsm-12.txt>, Huawei Technologies (USA), Jacobs University Bremen, February 2008.
- [13] D. Harrington. Transport Security Model for SNMP. Internet Draft (work in progress) <draft-ietf-isms-transport-security-model-07.txt>, Huawei Technologies, November 2007.
- [14] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251, SSH Communications Security Corp, Cisco Systems, January 2006.
- [15] D. Harrington and J. Salowey. Secure Shell Transport Model for SNMP. Internet Draft (work in progress) <draft-ietf-isms-secshell-10.txt>, Futurewei Technologies, Cisco Systems, February 2008.
- [16] U. Blumenthal and B. Wijnen. User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3). RFC 3414, Lucent Technologies, December 2002.
- [17] R. Frye, D. Levi, S. Routhier, and B. Wijnen. Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework. RFC 3584, Vibrant Solutions, Nortel Networks, Wind River Systems, Lucent Technologies, August 2003.
- [18] V. Marinov and J. Schönwälder. Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP). Internet Draft (work in progress) <draft-marinov-isms-tlstm-00.txt>, Jacobs University Bremen, February 2007.
- [19] A. G. Morgan. The Linux-PAM Application Developers' Guide. Technical report, November 1999.
- [20] J. Schönwälder. Simple Network Management Protocol (SNMP) Context EngineID Discovery. Internet Draft (work in progress) <draft-ietf-opsawg-snmp-engineid-discovery-02.txt>, Jacobs University Bremen, February 2008.
- [21] M. Björklund. YANG - A data modeling language for NETCONF. Internet Draft (work in progress) <draft-ietf-netmod-yang-02.txt>, Tail-f Systems, November 2008.
- [22] J. Schönwälder. Common YANG Data Types. Internet Draft (work in progress) <draft-ietf-netmod-yang-types-01.txt>, Jacobs University, November 2008.
- [23] K. McCloghrie and F. Kastenholz. The Interfaces Group MIB. RFC 2863, Cisco Systems, Argon Networks, June 2000.
- [24] Pastry - a scalable, decentralized, self-organizing and fault-tolerant substrate for peer-to-peer applications, 2007. <http://freepastry.org>.
- [25] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM 2001*, August 2001.

- [26] *Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications*, August 2001.
- [27] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418, New York, NY, USA, 2003. ACM.
- [28] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. in *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, March 2002.
- [29] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in p2p systems. *Communications of the ACM*, 46(2):43–48, 2003.
- [30] Thomas Bocek. TomP2p - a distributed multi map. <http://www.csg.uzh.ch/publications/software/TomP2P/>.
- [31] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2616, United States, 1999.
- [32] J. Schönwälder. SNMP Traffic Measurements and Trace Exchange Formats. RFC 5345, Jacobs University Bremen, October 2008.
- [33] J. Schönwälder, A. Pras, M. Harvan, J. Schippers, and R. van de Meent. SNMP Traffic Analysis: Approaches, Tools, and First Results. In *Proc. 10th IFIP/IEEE International Symposium on Integrated Network Management (IM 2007)*, pages 323–332, May 2007.
- [34] J.G. van den Broek, J. Schönwälder, A. Pras, and M. Harvan. SNMP Trace Analysis Definitions. In *Proc. of the 2nd International Conference on Autonomous Infrastructure, Management and Security (AIMS 2008)*, number 5127 in LNCS. Springer, July 2008.

Appendix 1: Software packaging call one text

EMANICS Work Package 6 : Open Source Software Packaging and Tutoring

This call covers the task T6.2 in the first 9 months of phase 2 : july 2007 - march 2008 It has a overall maximum budget of 9 K Euros.

Proposal Sheet

The proposal has to be filled and sent to the WP Leader : Olivier Festor, Olivier.Festor@loria.fr) before August 30th 12 AM. The flat funding for Packaging and Tutoring will be of 3K per initiative.

Every supported initiative commits to provide 1 months before the end of the 9 months supporting period, a detailed description of the addressed software (2-4 pages) together with a precise presentation of the packaging efforts made (2-3 pages) to be included in the deliverable of the project. This is to be provided in the WP6 deliverable format which in Phase 2 will be LATEX.

Note: fill the proposal carefully and provide detailed, precise and measureable commitments. Incomplete proposals will be immediately rejected.

=====

0. Proposal Title:

1. Addressed Software

name and short description of the existing Open Source project addressed

2. Detailed List of Packaging and Documentation Activity planed under the support of EMANICS

(describe all tasks planed and extensions envisioned as part of this support)

3. Expected Impact

(what new markets the enhanced software will conquer, how many distributions are envisioned, where is it going to be integrated, what visibility the NoE can gain through this support)

Appendix 2: Software development call one text

EMANICS Work Package 6 : Open Source Software Initiatives

This call covers the first 9 months of phase 2 : july 2007 - march 2008)

It has a overall maximum budget of 45K Euros.

Proposal Sheet

The proposal has to be filled and sent to the WP Leader : Olivier Festor, Olivier.Festor@loria.fr) before August 30th 12 AM. Cooperative Open Source developments will be favored over proposals including a single participant. The budget for Open Source overall was reduced in phase 2. In order to fund several proposals, collaborative requests should not exceed 15K. Single request should not exceed 7.5K/request.

Every supported initiative commits to provide 1 months before the end of the 9 months supporting period, a detailed description of the software (5-10 pages), a precise presentation of the made changes (2-3 pages), and an impact evaluation (1-2 pages) to be included in the deliverable of the project. This is to be provided in the WP6 deliverable format which in Phase 2 will be LATEX.

Note: fill the proposal carefully and provide detailed, precise and measureable commitments. Incomplete proposals will be immediately rejected.

=====

0. Proposal Title:

1. Overall Open Source Software Description

(A 1/2 to 3/4 page description of the concerned software, its current status, its visibility, its use, ... In case of a non existing soft the emphasis should be made on its need and its impact on the community)

2. Licensing and distribution scheme

(license type and distribution scheme used for the software : GPL, LGPL, QPL,; available on a given forge, is it or will it be embedded in a third party software distribution, ...)

3. Detailed List of Extensions Planed under the support of EMANICS

(describe all tasks planed and extensions envisioned as part of this support)

4. Expected Impact

(what new markets the enhanced software will conquer, how many

distributions are envisioned, where is it going to be integrated, what visibility the NoE can gain through this support ?

5. Cooperation level

(which parts of the extensions planned come from a cooperation among one or more partners in EMANICS, e.g. X will integrate in his software the algorithm defined by Y).

6. Cost and Requested support

(Expected cost overall + requested support) Cost includes the resources the partner puts on the development without being supported by the NoE. These efforts must be measurable at the end of the funding period. Request Support contains the amount of money asked to the NoE. The requested support should precisely specify how it is distributed among salary and equipment.

Appendix 3: Software development call two text

EMANICS Work Package 6 : Open Source Software Initiatives

This call covers the last 9 months of phase 2 : April December 2008

It has a overall maximum budget of 50K Euros.

Proposal Sheet

The proposal has to be filled & sent to the WP Leader : Olivier Festor, Olivier.Festor@loria.fr) before April 21th 12 AM. Cooperative Open Source developments will be favored over proposals including a single participant. The budget for Open Source overall was reduced in phase 2. In order to fund several proposals, collaborative requests should not exceed 15K. Single request should not exceed 7.5K/request.

Every supported initiative commits to provide 1 months before the end of the 9 months supporting period, a detailed description of the software (5-10 pages), a precise presentation of the made changes (2-3 pages), and an impact evaluation (1-2 pages) to be included in the deliverable of the project. This is to be provided in the WP6 deliverable format which in Phase 2 will be LATEX.

Note: fill the proposal carefully and provide detailed, precise and measureable commitments. Incomplete proposals will be immediately rejected.

=====

0. Proposal Title:

1. Overall Open Source Software Description

(A 1/2 to 3/4 page description of the concerned software, its current status, its visibility, its use, ... In case of a non existing soft the emphasis should be made on its need and its impact on the community)

2. Licensing & distribution scheme

(license type & distribution scheme used for the software : GPL, LGPL, QPL,; available on a given forge, is it or will it be embedded in a third party software distribution, ..., ...)

3. Detailed List of Extensions Planed under the support of EMANICS

(describe all tasks planed and extensions envisioned as part of this support)

4. Expected Impact

(what new "markets" the enhanced software will "conquer", how many

distributions are envisioned, where is it going to be integrated, what visibility the NoE can gain through this support ?

5. Cooperation level

(which parts of the extensions planned come from a cooperation among one or more partners in EMANICS, e.g. X will integrate in his software the algorithm defined by Y).

6. Cost & Requested support

(Expected cost overall + requested support) Cost includes the resources the partner puts on the development without being supported by the NoE. These efforts must be measurable at the end of the funding period. Request Support contains the amount of money asked to the NoE. The requested support should precisely specify how it is distributed among salary & equipment.

Appendix 4: Software packaging call two text

EMANICS Work Package 6 : Open Source Software Packaging & Tutoring

EMANICS Work Package 6 : Open Source Packaging

This call covers the last 9 months of phase 2 : April December 2008

It has a overall maximum budget of 9K Euros.

Proposal Sheet

The proposal has to be filled & sent to the WP Leader : Olivier Festor, Olivier.Festor@loria.fr) before August 30th 12 AM. The flat funding for Packaging & Tutoring will be of 3K per initiative.

Every supported initiative commits to provide 1 months before the end of the 9 months supporting period, a detailed description of the addressed software (2-4 pages) together with a precise presentation of the packaging efforts made (2-3 pages) to be included in the deliverable of the project. This is to be provided in the WP6 deliverable format which in Phase 2 will be LATEX.

Note: fill the proposal carefully and provide detailed, precise and measureable commitments. Incomplete proposals will be immediately rejected.

=====

0. Proposal Title:

1. Addressed Software

name & short description of the existing Open Source project addressed

3. Detailed List of Packaging & Documentation Activity planed under the support of EMANICS

(describe all tasks planed and extensions envisioned as part of this support)

4. Expected Impact

(what new "markets" the enhanced software will "conquer", how many distributions are envisioned, where is it going to be integrated, what visibility the NoE can gain through this support ?